Cryptanalysis of Some Lightweight Symmetric Ciphers

Abdelraheem, Mohamed Ahmed Awadelkareem Mohamed Ahmed

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Cryptanalysis of Some Lightweight Symmetric Ciphers

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at the

Department of Mathematics
in
The Technical University of Denmark

by

Mohamed Ahmed A. M. A. Abdelraheem

December 2012

To my

mother, father and brothers

with love

Title of Thesis:

Cryptanalysis of Some Lightweight Symmetric Ciphers

PhD Student:

Mohamed Ahmed Abdelraheem

Assessment Committee:

Associate Professor Christian Rechberger (DTU Compute, Denmark)

Professor Thomas Johansson (Lund University, Sweden)

Senior Researcher Anne Canteaut (INRIA Paris-Rocquencourt, France)

Abstract

In recent years, the need for lightweight encryption systems has been increasing as many applications use RFID and sensor networks which have a very low computational power and thus incapable of performing standard cryptographic operations. In response to this problem, the cryptographic community designed a number of lightweight cryptographic primitives that varies from stream ciphers, block ciphers and recently to hash functions. Out of these many lightweight primitives, the block cipher PRESENT gets a lot of attention from the cryptographic community and it has been recently adopted by ISO as one of the international standards in lightweight cryptography.

This thesis aims at analyzing and evaluating the security of some the recently proposed lightweight symmetric ciphers with a focus on PRESENT-like ciphers, namely, the block cipher PRESENT and the block cipher PRINTcipher.

We provide an approach to estimate the probability of differential and linear approximations with low-weight differential and linear characteristics on PRESENT-like ciphers as well as ciphers allowing low hamming weight differential and linear characteristics. We study the effect of key scheduling in the distribution of linear approximations on a variant of PRESENT with identical round keys. We propose a new attack named the Invariant Subspace Attack that was specifically mounted against the lightweight block cipher PRINTcipher. Furthermore, we mount several attacks on a recently proposed stream cipher called A2U2.

Abstrakt (in Danish)

I de seneste år er efterspørgslen efter systemer til letvægtskryptering steget, da mange anvendelser involverer RFID og sensornetværk, som af natur har meget lav ydeevne, og derfor er ude af stand til at udføre sædvanlige kryptografiske operationer. For at imødekomme dette problem, har det kryptografiske fællesskab udviklet adskillige byggeblokke til brug i letvægtskryptering, som indebærer strømcifre, blokcifre og, som det seneste, hashfunktioner. Ud af disse letvægtsalgoritmer, får især blokcifret PRESENT megen opmærksomhed fra det kryptografiske fællesskab, og er for nylig blevet vedtaget af ISO som en international standard i letvægtskryptografi.

Denne afhandling har til formål at analysere og evaluere sikkerheden af nogle af de senest foreslåede symmetriske metoder til letvægtskryptering, med fokus på cifre som designmæssigt ligner PRESENT, navnlig PRESENT selv, samt blokcifret PRINTcipher.

Vi giver en metode til at estimere sandsynligheden af differential- og lineære approksimationer, for karakteristikker af lav vægt, på kryptosystemer som designmæssigt ligner PRESENT, såvel som systemer der tillader differential- og lineære karakteristikker af lav Hamming vægt. Vi undersøger virkningen af nøgleskema-planlægningen på fordelingen af lineære approksimationer for en variant af PRESENT med identiske rundenøgler. Vi foreslår et nyt angreb med navnet *Invariant underrum-angreb*, som specifikt blev anvendt på letvægtscifret PRINTcipher. Desuden giver vi adskillige angreb på et nyt strømciffer kaldet A2U2.

Preface

Lightweight cryptography deals with the design of cryptographic primitives that are suitable to fit and run on small hardware devices such as RFID tags, sensor networks and contactless smart cards. As the standard symmetric algorithms such as AES cannot fit in these low cost small devices. The cryptographic community has recently responded by designing a number of lightweight symmetric primitives suitable for constrained environments. This thesis evaluates the security of some of the recently proposed symmetric-key ciphers. This thesis is organized as follows.

Chapter 1 gives a brief introduction about cryptography and cryptanalysis. We highlight the goals and services of cryptography and the goals of a cryptanalyst and define some of the attack models used by cryptanalysts. We also describe some generic attacks on symmetric-key ciphers.

Chapter 2 describes the common cryptanalytic techniques used in symmetric-key ciphers. We describe the most versatile attacks on symmetric-key ciphers, namely, differential and linear cryptanalysis and some improvements of these attacks. We also give a short description about other attack variants such as the slide attack and algebraic attacks.

Chapter 3 provides a brief introduction about symmetric-key lightweight cryptography. We describe the lightweight ciphers that are analyzed in this thesis, namely, the block cipher PRESENT, the block cipher PRINTcipher and the stream cipher A2U2.

Chapter 4 addresses two subjects on differential and linear cryptanalysis in PRESENT-like ciphers. The first subject concerns the estimation of the probabilities of low-weight differential and linear approximations. The main result of this subject is in the extension a previously known method to find better estimations for the correlation on linear approximations on the block cipher PRESENT and the core permutation of SPONGENT. This result was published in ICISC 2012 [1]. The second subject concerns the effect of key scheduling on the distribution of differential and linear approximation on PRESENT-like ciphers. The

main result of this subject is in showing that variance of the distribution of some linear approximations in PRESENT with identical keys is significantly larger than that of PRESENT with independent round keys. This result was part of a work about the distribution of linear biases published in CRYPTO 2012 [2]. Note that [2] was a joint work with Martin Ågren, Peter Beelen and Gregor Leander.

Chapter 5 is concerned about the differential cryptanalysis of PRINTcipher. The key-dependent linear layer of PRINTcipher seems to complicate differential cryptanalysis. We show that is the not the case by providing two differential attacks on reduced rounds of PRINTcipher. One of these attacks recovers the key-dependent linear layer by computing the rth root of a permutation in the symmetric group S_{48}. This work was published at FSE 2011 [4]. Note that this was a joint work with Gregor Leander and Erik Zenner. The chapter also shows how to design an authentication protocol whose security is based on a multi-valued function, namely, the r-th roots of a permutation in the symmetric group S_n.

Chapter 6 provides a new attack on block ciphers called the Invariant Subspace Attack. The attack was successfully applied to PRINTcipher and it managed to break the whole PRINTcipher for a certain number of keys. We show that this attack can be seen as a weak key variant of the statistical saturation attack previously applied on the block cipher PRESENT. This work was published at CRYPTO 2011 [83]. Note that this was a joint work with Hoda AlKhzaimi, Gregor Leander and Erik Zenner.

Chapter 7 provides several attacks on the stream cipher A2U2. The first attack uses only two chosen plaintexts to break the cipher in a second. The second attack uses a guess-and-determine approach to mount an attack with time complexity 2^{49}. The third attack is a chosen IV attack that recovers 5-bit of the secret key used to initialize the LFSR counter. The fourth attack is a known plaintext attack that targets the low number of initialization rounds to recover the secret key with time complexity 2^{38}. Finally, an attack that exploits the noisy key stream is proposed. This work was published at IMACC 2011 [3]. Note that this was joint work with Julia Borghoff, Erik Zenner and Mathieu David.

Chapter 8 wraps up the results of this thesis and suggests some topics for future work.

The work presented in this thesis was performed at the Department of Mathematics in the Technical University of Denmark. The author was supervised by Professor Lars Knudsen and co-supervised by Associate Professor Gregor Leander and Associate Professor Erik Zenner. The author was funded by a scholarship from DTU Mathematics.

During the three years of the PhD studies, the following papers were published

- Mohamed Ahmed Abdelraheem. Estimating the probabilities of low-weight differential and linear approximations on PRESENT-like ciphers. In Taekyoung Kwon, Mun-Kyu Lee, Daesung Kwon, editors, *ICISC*, 2012. to appear.

- Mohamed Ahmed Abdelraheem, Céline Blondeau, Maria Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of ARMADILLO2. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT, volume 7073 of Lecture Notes of Computer Science*, pages 308-326. Springer, 2011[1].

- Mohamed Ahmed Abdelraheem, Martin Ågren, Peter Beelen, and Gregor Leander. On the distribution of linear biases: Three instructive examples. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO, volume 7417 of Lecture Notes in Computer Science*, pages 50-67. Springer, 2012.

- Mohamed Ahmed Abdelraheem, Julia Borghoff, Erik Zenner, and Mathieu David. Cryptanalysis of the light-weight cipher A2U2. In Liqun Chen, editor, *IMA Int. Conf., volume 7089 of Lecture Notes in Computer Science*, pages 375-390. Springer, 2011.

- Mohamed Ahmed Abdelraheem, Gregor Leander, and Erik Zenner. Differential cryptanalysis of round-reduced PRINTCIPHER: Computing roots of permutations. In Antoine Joux, editor, *FSE, volume 6733 of of Lecture Notes in Computer Science.* pages 1-17. Springer, 2011.

- Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda Alkhzaimi, Erik Zenner. A Cryptanalysis of PRINTCIPHER: The Invariant Subspace Attack. In Phillip Rogaway, editor, *CRYPTO, volume 6841 of Lecture Notes in Computer Science*, pages 206-221. Springer, 2011.

[1]Note that this paper is not included in this thesis

Acknowledgments

First and Foremost I would like to thank ALLAH — the Ever-Living and the Sustainer of all existence, the One that neither begets nor is born and nor is there to Him any equivalent — unlimited thanks that are suitable for His majesty and His perfect attributes, for His uncountable blessings that He has bestowed upon me — one bless is surrounding me with all the people that made this thesis possible. The first of all those people are my parents, Fatima and Awadelkareem and my brothers. This thesis would not have been possible without your prayers, encouragements and constant support all the time.

I would like to thank my supervisor Lars Knudsen for giving me the opportunity to do a PhD at his group, for his advice, for the open door policy and for many useful feedbacks during the writing of this thesis. I believe that your supervision has allowed me to grow up as a researcher.

This thesis would not have been possible without my co-supervisor Gregor Leander. Thank you for the many ideas, for the open door policy, for useful comments on my work and for improving my presentation and writing skills.

Many thanks go to Erik Zenner for being my co-supervisor during my the first and half year of my PhD study, for all the weekly and fruitful meetings that we had, for the open-door policy, for improving my presentation and writing skills. Words fail to express my appreciation to your supervision.

I am very grateful to Tom Høholdt for being careful about my life in Denmark and my PhD progress while and after he was the head of PhD at DTU Mathematics.

I would like to thank the rest of DTU Crypto group — Andrey, Christian, Christiane, Huda, Julia, Martin Albrecht, Martin Lauridsen and the former colleagues Krystian, Praveen, Søren, Valérie and also to the group visitors Anne and Reza — for broadening my knowledge on cryptology and for all the nice and useful discussions that I encountered with all of you. Thank you Julia, for being a co-author and for proof-reading part of this thesis. Thank you Martin for the Danish translation of the abstract of this thesis. Thank you Huda, for being my office mate and for the all useful discussions that we had.

It has also been a pleasure to be part of the Discrete Mathematics group. I would like to thank

those I have not mentioned: Carsten, Peter, Johan, Fernando, Nhut, Safia and Seongmin. Special thanks go to Johan for being my office mate for almost one year. Thank you for all the useful discussions that we had. I learned a lot from you about Denmark, Linux, Open Source and Programming.

Special thanks to my colleague Kun Marhadi for being a good friend and for the support and all the nice discussions that we had.

I am really privileged to be part of DTU Mathematics during the past three years. Thanks to all the staff of DTU Mathematics for creating such a nice atmosphere and friendly environment. Many thanks to all the current and former PhD colleagues at DTU Mathematics with whom I shared many nice and useful discussions during lunches and PhD trips.

I would like to thank the administrative staff at DTU Mathematics for all the help they provided whenever I needed. Thank you to Anna, Dorte, Hell Vibeke, Kari, Poul-Erik, Ulla and Wanja. Special thanks go to Wanja for helping me in filling the Travel Management System whenever I come from a conference or course abroad.

I would like to thank Vincent Rijmen for hosting me at the COSIC group in KU Leuven. Thank you for giving me this opportunity. Many thanks go to all the people at the COSIC group for the useful discussions that I had during my time there. Special thanks go to Nicky Mouha for being my mentor during my stay there and to Andrey — who was then with COSIC — for all the useful discussions that we had.

Many thanks go to all those who helped or offered to help me while waiting for a re-entry VISA to Denmark after the loss of all my identity cards during my holidays in Saudia Arabia. In particular, I thank Lars and my friends Ammar Arabi, Mohammed Alfaki, Ayman Basheer, Hassan Ibrahim, Tarig Mahdi and Hazzaa. Special thanks go to my uncle Usama, my cousin Atif and my brother Ashraf for helping me with all the procedures during this period from the very beginning till I arrived back to Denmark. Thank you Atif for that nice Lemon mixed with fresh Mint juice which has become since then my favourite home-made juice. Special thanks also go to my dear friend Hazzaa for all the help he provided when I was in need — A friend in need is a friend indeed. Thank you also for all the nice and many useful discussions that we had.

I would like to thank the Sudanese community in Copenhagen. In particular, I thank Ahmed, Ammar, Amir, Haitham, Hamid and Hazzaa. I am very grateful to Ahmed and Haitham for all the help they provided during my first days in Denmark. Special thanks go to Amir and his family for the many dinner invitations that I enjoyed at their home.

Table of Contents

Chapter 1

Introduction

Cryptography is the art and science of designing security algorithms to provide certain security services. *Cryptanalysis* is the art and science of analyzing and defeating the security claims of these algorithms. The branch of science that embodies both cryptography and cryptanalysis is called *Cryptology*. The main goal of cryptography is to secure communications between two parties (sender and receiver) by transforming a message (*plaintext*) to a scrambled message (*ciphertext*) using a secret key. The process of transforming a plaintext into a ciphertext is called *encryption* and the process of unscrambling the ciphertext to recover the original plaintext is called *decryption*.

Before the digital era, cryptography was used mainly by governments to secure their communication channels and thus it was all about providing the service of *confidentiality* using encryption-decryption methods. Nowadays, cryptography is widely used to solve real-world information security problems that arise in many digital applications such as ATM cards, computer passwords and online shopping. Thus in addition to confidentiality, modern cryptography provides the following services to solve various aspects of information security problems:

- *Authentication* allows the receiver to verify the sender's identity.
- *Data Integrity* enables the receiver to verify that the sender's message has not been modified.
- *Non-repudiation* prevents the sender from denying any message he previously sent.

1.1 Cryptographic Primitives

The mathematical functions or algorithms that provides the above mentioned services are called *cryptographic primitives*. These primitives include the following three schemes,

but are not limited to them: *symmetric-key ciphers*, *public-key ciphers* and *cryptographic hash functions*.

1.1.1 Symmetric-Key Ciphers

Symmetric-key ciphers use one key for encryption and decryption between two parties. Thus, they require that the two communicated parties agreed beforehand on a key. The whole security of symmetric-key primitives depend on the secrecy of the key; revealing the key means that encryption and decryption are possible by anyone. Symmetric-key ciphers are used to provide the services of confidentiality and authentication and they can be divided into two schemes. One scheme operates on a block of bits and these kind of ciphers are called *block ciphers* and the other scheme generates a pseudo-random stream of bits and mixes it with the plaintext bits and these kind of ciphers are called *stream ciphers*. In the following, we give a brief overview about block ciphers and stream ciphers.

Block Ciphers

A block cipher divides the input (plaintext) into block of bits of the same length and then encrypts each block by a secret key. The output of the encryption operation must be invertible using the same secret key used in the encryption operation. A block cipher with a block size n takes a block of plaintext represented in bits $(m_0, \cdots, m_{n-1})$ and outputs a binary vector $(c_0, \cdots, c_{n-1})$ depending on the secret key. More formally, a block cipher is defined as follows.

Definition 1. *A block cipher with a block size of n bits and key size of k bits is an invertible mapping $F : \mathcal{M} \times \mathcal{K} \to C$, where $\mathcal{M} \in F_2^n$ is the message space, $\mathcal{K} \in F_2^k$ and $\mathcal{C} \in F_2^n$.*

A block cipher is a family of n-bit permutations of size 2^k where each secret key yields one permutation out of the $2^n!$ n-bit permutations. In order to pick a totally random permutation or an ideal permutation from the set of all n-bit permutations, we would need a key of size $\log_2(2^n!) \approx (n-1)2^n$ bits. This is a huge number and would be impractical to have such key size, so the design goal would be to approximate ideal ciphers in practice by drawing 2^k permutations uniformly at random from the set of all n-bit permutations.

Modern block ciphers are iterated ciphers. They iterate a round function a number of times. The round function consists of a linear and nonlinear layers which transform the given state (the plaintext in the first iteration) using a round subkey. There are two construction types of block ciphers:

- *Feistel Cipher* operates on half of the state, i.e., the state at round i is divided into two equal halves (L_i, R_i). The round function F is applied to one half, say R_i, using a round subkey k_i and then its output is xored with the other half. Then the two halves are exchanged with each other. Thus $L_{i+1} = R_i$ and $R_{i+1} = L_i \oplus F(R_i, K_i)$.
- *Substitution Permutation Network* (SPN) operates on the whole state. They transform the state using a substitution layer followed by a permutation layer. The substitution layer is a nonlinear layer consisting of small Sboxes which are vectorial boolean functions that substitutes a small vector (usually ≤ 8 bits) of input bits with another small vector of output bits (usually ≤ 8 bits). The permutation layer applies a linear transformation to the output of substitution layer. The simplest linear layer is the bit-wise permutation which permutes the state of the bits but there are complex linear layers that performs matrix multiplication.

Examples for notable block ciphers are the DES, a Feistel cipher and was the previous encryption standard and the AES which is an SPN design and the current encryption standard.

Stream Ciphers

The only provably secure cipher, called *one time pad*, is a classical stream cipher whose secret key is used only once and has the same length as the plaintext under encryption. The cipher simply xors the plaintext bits with the secret key bits. Obviously this encryption scheme has practical problems. Modern stream ciphers tries to emulate the action of this provably secure scheme by using a short secret key and mixing it with a random initial value in order to generate a very long pseudo-random key stream sequence (this is called the initialization phase) which is then xored with the given plaintext to produce the ciphertext.

Modern stream ciphers are usually constructed using Linear Feedback Shift Registers (LFSR) and Non-Linear Feedback Shift Registers (NFSR). Linear Feedback Shift

Registers devices are used to produce long period sequences from short ones. An LFSR of length n consists of n stages and is associated to a connection polynomial, $c_nX^n + c_{n-1}X^{n-1} + \cdots + c_1X + c_0$, where $c_i \in \mathbb{F}_2$ that is used to update its state. Each stage stores one bit (or word) and has one input and one output. The flow of bits is controlled by a clock. At each clock tick, the contents of stage 0 is the output and forms part of the output sequence. Stage i is moved to stage $i-1$ for each $1 \leq i \leq n-1$. Stage $n-1$ is filled by the feedback bit(or word), s_j, which is formed by xoring together a fixed subset of the previous stages $(0, 1, \ldots, n-1)$ depending on c_i's, the coefficients of the connection polynomial,

$$s_j = \bigoplus_{i=1}^{n} c_i s_{j-i}.$$

An LFSR of length produces a maximal sequence of length $2^n - 1$ iff its connection polynomial is primitive. LFSRs are vulnerable to the powerful $O(n^2)$ linear complexity Berlekamp-Massey attack [88] which requires only $2n$ consecutive sequence bits (or words) to deduce the c_i's. Therefore, the LFSR is used together with a nonlinear Boolean function to avoid the Berlekamp-Massey attack. Another component that is also used in modern stream ciphers is the NFSR which works similar to LFSR except that its feedback function is a nonlinear Boolean function of the state. Examples for NFSR-based stream ciphers are the lightweight ciphers Grain [60] and Trivium [32].

1.1.2 Public-Key Ciphers

Also known as Asymmetric-key ciphers. They use one key for encryption called "*public key*" and another key for decryption called "*private key*". The public key is known by everyone whereas the private key is kept secret by its user and it is never possible to deduce the private key from the public key. A public-key cipher can be defined as a function that maps a plaintext message to a ciphertext message and can be computed by anyone having the public key but its inverse is infeasible to compute unless we have the private key. This one way encryption function is often called a trapdoor function. The hardness of inverting these functions relies on the fact that there are some mathematical problems that there are no known algorithms to solve them in a reasonable amount of time such as the Integer Factorization Problem and the Discrete Logarithm Problem. The most prominent public-key ciphers are RSA and ElGamal. Public-key ciphers are used to provide the services of confidentiality, authentication, non-repudiation and secret key establishment.

1.1.3 Cryptographic Hash Functions

A cryptographic hash function maps a binary message of any arbitrary length to a small binary message of a fixed length often called the *hash value* or the *message digest*. The message digest serves as a small unique representation and a digital fingerprint of the actual lengthy message. Since the input size of the hash function is larger than its output size, many messages will have the same message digest but finding such messages should be infeasible. A good cryptographic hash function must be easy to compute and must possess the following properties:

- *Collision resistance:* It is infeasible to find two messages that have the same hash value.
- *Preimage resistance:* Given a hash value, it is infeasible to generate its actual message.
- *Second preimage resistance:* Given a specific message, it is infeasible to find another different message that has the same hash value.

Cryptographic hash functions are used in many security applications to provide the services of data integrity and non-repudiation when used with a public-key cipher. Another different primitive but usually constructed from either a keyed hash function or a block cipher is called Message Authentication Code (MAC). It has different security properties and it is used to provide authentication.

The focus of this thesis is in the cryptanalysis of symmetric-key ciphers, so the next section specifically discusses cryptanalysis of symmetric-key ciphers and more specifically block ciphers.

1.2 Cryptanalysis

In this section, we describe the cryptanalyst's (aka attacker or adversary) goals and the attack models based on the data available to him. We also discuss the measurements of the complexity of a cryptanalytic attack. We close the section by describing some generic attacks on block ciphers.

1.2.1 Attacker's Goals

The attacker's goal is to find the secret key but he can also tries to achieve other useful information. In [75, 79], Knudsen classified various *types of attacks* on block ciphers based on the attacker's gained information:

- *Total break*: An attacker finds the secret key of the cipher.
- *Global deduction*: An attacker finds equivalent algorithms for performing encryption and decryption without knowing the secret key.
- *Local deduction*: An attacker finds the plaintext of a new given ciphertext.
- *Distinguishing algorithm*: An attacker is able to distinguish the block cipher from a randomly chosen permutation.

1.2.2 Attack Models and Complexity

Cryptanalytic attacks can be divided based on the amount of data available. First, the attacker is assumed to know the cryptographic algorithm under attack. This is known in the literature as Kerckhoff's principle. The following represents some of the *attack models* used in symmetric key cryptanalysis based on the data available to the attacker:

- *Ciphertext only*: the attacker only knows the ciphertexts
- *Known plaintext*: the attacker knows some plaintexts with their corresponding ciphertexts
- *Chosen plaintext (ciphertext)*: the attacker can choose his own set of plaintexts (ciphertexts) and obtain their corresponding ciphertexts (plaintexts), i.e. the attacker has access to the encryption (decryption) device.
- *Adaptive chosen plaintext (ciphertext)*: similar to the chosen plaintext (ciphertext) attack except that here the attacker base his next choice of plaintext (ciphertext) on his observations from the previous encryptions.
- *Related key attacks*: here the attacker knows some plaintexts-ciphertext obtained under the encryption of at least two unknown keys (k_1, k_2) but they have a known or chosen relationship (i.e. $k_2 = f(k_1)$ where f is known or chosen).

The *complexity* of an attack is determined by the dominant complexity among the following three different complexity measures together with the *success probability* of the attack:

- *Time complexity*: the processing time taken to mount the attack.
- *Data complexity*: the expected amount of data (plaintext-ciphertext) for the attack to succeed.
- *Memory complexity*: the memory size required to process the attack.

1.2.3 Generic Attacks

Generic attacks are the kind of attacks that are possible regardless of the internal structure of the target cipher. The most obvious generic attack is the *brute force attack* or *exhaustive key search* which is simply trying all the possible keys. Another generic attack is called *table lookup* and this attack builds up a huge table that is used later to find the key instantly. Another generic attack that combines both the exhaustive key search and the table look-up attack is called the *time-memory tradeoff*. We conclude our discussion on generic attacks by describing meet-in-the-middle attacks especially the attack on 2DES. While the meet-in-the-middle attack on 2DES is a generic attack, advanced meet-in-the-middle attacks do exploit the internal structure of the cipher and more specifically they exploit some weaknesses in its key schedule. In the following we discuss these four generic attacks.

Brute Force:

Suppose we have an n-bit block cipher E_k with a k-bit key size. Then a brute force can be performed either by using a ciphertext only where the attacker searches throughout the key space until a meaningful plaintext is found or by using a known plaintext/ciphertext pair where the attacker looks for the key that yields the plaintext/ciphertext pair. We might possibly get more than one key candidate for the target secret key when known the key size k is larger than the block size n. This is due to the fact that the probability of encrypting an n-bit plaintext (m) to an n-bit ciphertext (c) under one key is 2^{-n}. Thus the expected number of keys that encrypts m to c is 2^{k-n}. Thus in order to get a unique key we need to test $\lceil \frac{k}{n} \rceil$ plaintext/ciphertext pairs.

If we only need one plaintext/ciphertext, then the worst case time complexity will be 2^k encryptions of the cipher under consideration. If the secret key lies in the first or last half of the key space, then the time complexity will be 2^{k-1} encryptions. Now the target secret key is chosen at random and with probability $\frac{1}{2^k}$ and each k_i takes i encryptions. Therefore, the average time complexity can be computed as follows

$$\sum_{i=1}^{2^k} i \cdot \Pr(\text{secret key} = k_i) = \sum_{i=1}^{2^k} \frac{i}{2^k} \approx 2^{k-1}$$

Thus on average we need 2^{k-1} encryptions. Parallel computing is usually used to speedup the exhaustive search. For example, if the amount of computing power available to the attacker is P, then the time taken for a brute force attack will be in the order of $\approx \frac{2^k}{P}$. Therefore, it is recommended that when designing a cipher, that the secret key length k is chosen such that $\frac{2^k}{P}$ is larger than the life time of the data protected by the cipher [66].

Table Look-Up

Rather than carrying out the exhaustive search attack everytime. The table look-up attack builds a huge table once using exhaustive search in order to be used later to find the key instantly. The table look-up attack consists of two phases: one phase called the offline phase where exhaustive key search is used to build a table that maintains 2^k entries where each entry consists of a specified plaintext m that is likely to be sent (plaintext is a common word) and an encryption key together with the corresponding ciphertext. The other phase called the online phase where we attack the cipher under consideration by getting a ciphertext c obtained from encrypting the likely specified plaintext at the offline phase and then finding the target key by looking up for c in the table built in the offline phase.

Time-Memory Tradeoff:

Clearly the exhaustive key search attack consumes time and negligible memory whereas the table lookup consumes memory and negligible time. This triggered and led up to the time-memory tradeoff attack developed by Hellman in [61]. The time-memory tradeoff attack is a probabilistic method and its success probability depends on time and memory that are allocated by the attacker. Likewise the table look-up attack, the time-memory

tradeoff attack consists of two phases: an offline phase where the attacker builds a table and stores it in memory in order to be used in the online phase to reduce the amount of time to recover a secret key.

In order to perform a time-memory tradeoff attack, we first choose a likely plaintext P and obtain its ciphertext $C = E_K(P)$, where E is an n-bit block cipher with key size k-bit and for simplicity let us assume first that $n = k$. Let K_0 be a possible key. Then we create a chain of t encryptions where our starting point denoted as SP equals K_0 and our ending point denoted as EP equals K_{t-1}. Now the chain is created as follows:

$$SP = K_0, \quad K_1 = E_{SP}(P), \quad K_2 = E_{K_1}(P), \quad \cdots \quad , \quad K_{t-1} = E_{K_{t-2}}(P) = EP$$

Now to cover more keys we create m chains where each key starts with a different starting point. Moreover we also assume that there is no overlap among the m chains. The m chains forms the following $m \times t$ matrix, where each row is created after performing $(t-1)$ encryptions.

$$\begin{array}{ccccc}
SP_1 = K_{1,1} & K_{1,2} = E_{SP_1}(P) & K_{1,3} = E_{K_{1,2}}(P) & \cdots & K_{1,t} = E_{K_{1,t-1}}(P) = EP_1 \\
SP_2 = K_{2,1} & K_{2,2} = E_{SP_2}(P) & K_{2,3} = E_{K_{2,2}}(P) & \cdots & K_{2,t} = E_{K_{2,t-1}}(P) = EP_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
SP_m = K_{m,1} & K_{m,2} = E_{SP_m}(P) & K_{m,3} = E_{K_{m,2}}(P) & \cdots & K_{m,t} = E_{K_{m,t-1}}(P) = EP_m
\end{array}$$

As we are assuming that none of the chains overlap, then our matrix covers mt distinct key values. To reduce memory requirements we only store the starting and ending points of each chain, i.e. (SP_i, EP_i) for all $1 \leq i \leq m$. Thus we reduce our memory storage from mt key values to $2m$ values at the cost of spending $(t-1)$ encryptions to recover each chain when we are looking for the secret key and thus we have a time-memory trade-off. Now let us explain the process of recovering the secret key. The attacker knows the ciphertext C of the chosen plaintext P used in the creation of the chains. The attacker performs the following encryption operations:

$$E_C(P) = \tilde{K}_1, E_{\tilde{K}_1}(P) = \tilde{K}_2, \cdots, E_{\tilde{K}_{t-1}}(P) = \tilde{K}_{t-2}$$

After each encryption step the attacker checks whether $\tilde{K}_i$ equals one of the m end points $(EP_1, \cdots, EP_m)$. Now assuming our chains cover all the key space, then we

must have

$$\tilde{K}_i = EP_j \quad \text{for some} \quad i \in \{1, 2, \cdots, t\} \quad \text{and} \quad j \in \{1, 2, \cdots, m\}$$

Now the attacker rebuilds the jth chain starting with SP_j and consequently one of the keys on this chain, say K_s must be equal to C. Thus $C = K_s = E_{K_{s-1}}(P)$ and therefore the secret key will be $K_{s-1} = E_{K_{s-2}}(P)$. Note that we have to avoid using C as a starting point when building the chains in the offline phase. In order for the chains to cover the whole key space we need to have $mt = 2^k$ and this is the time complexity of the offline phase. The online phase requires only $(t-1)$ encryptions. Since the assumption of having no overlaps among the chains is unrealistic and cannot be avoided, we will usually have a probabilistic attack anyway.

However, the overlaps in the above case where $n = k$ are fewer compared to the case $n > k$ which was in the original Hellman's time-memory attack [61] proposed on the DES cipher which has a block size, $n = 64$ bits and a key size, $k = 56$ bits. In order to create the chains represented in the above matrix, Hellman used what he called a *reduction function*, $f(K) = R(E_K(P))$, which simply truncates in some specific way the 64-bit of $E_K(P)$ to 56-bit so that it can be used as a key in order to produce the next value in the chain. The attack in this case is similar to the previous attack except here we have false positives or false alarms. For instance suppose that we found

$$f(C) = K_s = f(E_{K_{s-1}}(P)),$$

then unlike the $n = k$ case, the key K_{s-1} might not be the right key as the equality here holds for the truncated values and not for the actual values of C and $E_{K_{s-1}}(P)$. False alarms can be discarded by using them to decrypt C as the right key will always yield the plaintext P. The false alarms are a result of the overlap that is caused by the reduction function. Overlaps can happen in one chain or two chains can merge into one chain.

Therefore, the number of the distinct key values might be less than mt. The values of m and t decide the time-memory trade-off and the secret key is recovered with probability $\frac{mt}{2^k}$ under the assumption that all the mt key values are distinct. In other words, if the secret key exists in the chains, then it will be found by the time-memory tradeoff otherwise it will not be found. This makes the attacker aims at finding the suitable m and t values that increase the success probability of the time-memory trade-off attack.

Hellman showed that when choosing $m = t = 2^{\frac{k}{3}}$. The success probability will be $2^{-\frac{k}{3}}$ and to overcome this small probability Hellman proposed the use of around $l = 2^{\frac{k}{3}}$ different tables where each table has its own reduction function to avoid overlaps among the tables. If the first table doesn't yield the key, then the second table is tried, and so on until we find the key. Therefore, the total memory cost is $2ml = 2^{\frac{2k}{3}+1} \approx 2^{\frac{2k}{3}}$ and the total time complexity is at most $(t-1)l \approx 2^{\frac{2k}{3}}$ since in each table we perform $(t-1)$ encryptions and reductions. However, time-memory tradeoff attacks can be thwarted by adding some random value to any plaintext before encrypting it.

One notable improvement to Hellman's time-memory tradeoff is the work of Oechslin [92] which uses what he calls "rainbow table". This table uses $(t-1)$ successive reduction functions and thus different reduction functions within each point in a chain. For instance, all chains start with reduction function 1 and end with reduction function $(t-1)$. This reduces possible overlaps among the chains as any two chains that overlap will not merge unless they overlap at the same position or column in the table since different positions or columns in the table use different reduction functions. For more details about "rainbow tables", we refer the reader to [92].

Meet-In-The-Middle Attacks:

Meet-in-the-middle attacks were first developed by Diffie and Hellman to attack a version of DES called Double DES or 2DES which simply tries to solve the short length of the 56-bit DES key by increasing it to 112-bit through performing DES twice with two different keys, i.e. $2DES(K_1||K_2, P) = DES(K_2, DES(K_1, P)) = C$.

Given (P, C) the attack exploits the fact that $DES^{-1}(K_2, C) = DES(K_1, P)$. The meet in the middle attack on 2DES can described as follows:

1. To get a unique key we need to collect $\lceil \frac{112}{64} \rceil = 2$ (See the brute force attack) plaintext blocks and their corresponding ciphertexts (P_1, C_1) and (P_2, C_2).

2. For all the 2^{56} possible key values k_1 of K_1, compute $L = DES(k_1, P)$ and save L and k_1 in a hash table indexed by L.

3. After constructing the hash table, compute $R = DES^{-1}(k_2, C)$ for all the 2^{56} possible values k_2 of K_2 and check if there is as index in the hash table that equals to R. If there is no index that equals R in the hash table then $k_2 \neq K_2$ and we

continue another k_2. If there is an index L corresponding to the entry (L, k_1) in the hash table that equals R. Then $k_1||k_2$ is a possible candidate for $K_1||K_2$

4. If $k_1||k_2$ encrypts P_1 to C_1 and P_2 to C_2, then $k_1||k_2$ is probably the right key value.

The time complexity taken by the above algorithm is 2^{56} encryption operations taken at step 1 and at most 2^{56} decryption operations taken at step 3. Thus the total time complexity is 2^{57} DES encryption or decryption operations. The amount of memory taken by the hash table is $7 \times 8 \times 2^{56}$ bytes and this is approximately equivalent to 2^{22} TB which is a huge amount of memory (a time-memory tradeoff is possible here, for more information we refer the reader to [88]). However, this attack suggests that the security of 2DES is reduced from 112-bit key length to a 57-bit key length. This attack in 2DES led up to the suggestion of two triple DES or 3DES ciphers which use 3 iterations of DES. One is called 3DES3 and it uses three different keys and is defined as follows

$$3DES3(K_1||K_2||K_3, P) = DES(K_3, DES^{-1}(K_2, DES(K_1, P)))$$

The other one is called 3DES2 and it uses 2 different keys and is defined as follows

$$3DES2(K_1||K_2, P) = DES(K_2, DES^{-1}(K_1, DES(K_2, P)))$$

Here we note that DES^{-1} is used above so that 3DES and 3DES2 become easily compatible with DES since

$$DES(K, P) = 3DES3(K||K||K, P) = 3DES2(K||K, P)$$

3DES3 is obviously susceptible to a meet in the middle attack similar to the one on 2DES and its effective key length is clearly consists of the lengths of only two keys out of the three keys. Thus its effective key length is reduced from 168-bit (bits length of the three keys) to 112-bit (bits length of two keys). 3DES2 is suspectable to a meet-in-the-middle attack that takes time $2^{120-\log_2 t}$ and memory complexity in the order of t if t known plaintexts are available [88].

Chapter 2

Cryptanalysis Techniques

In his seminal paper [106], Claude Shannon mentioned two statistical properties that any cryptosystem should acquire in order to be secure against statistical analysis, namely *confusion* and *diffusion*. The *confusion* property could be described as having a complicated relation between the secret key and all the ciphertext bits in a way such that each ciphertext bit dependent on the entire secret key. A good confusion is accomplished if flipping one bit of a secret key, results in flipping several bits in the ciphertext. In cryptographic primitives this is often accomplished by the use of nonlinear vectorial Boolean functions aka substitution boxes (Sboxes). The *diffusion* property could be described as having the plaintext being distributed over the ciphertext in a way such that each plaintext bit affects as many bits as possible in the ciphertext. A good diffusion is accomplished if flipping one bit of a plaintext, results in flipping several bits in the ciphertext. In cryptographic primitives this is often accomplished by the use of linear mappings such as matrix multiplication. Most of the attacks on cryptographic primitives exploit weaknesses of these two important properties in the cryptosystem under consideration.

In this chapter, we discuss the two most powerful attacks agianst symmetric-key ciphers, namely, differential and linear cryptanalysis. We also discuss some of their extensions and close the chapter by giving a brief overview about the slide and algebraic attacks.

2.1 Differential Cryptanalysis

Differential Cryptanalysis is a chosen plaintext attack invented by Biham and Shamir [14]. The idea is to look into differences of plaintext pairs and their corresponding ciphertext pairs rather than looking into the values of a plaintext and its corresponding ciphertext. It exploits the existence of input differences whose corresponding output differences occur with high probability to deduce some information

about the secret key.

On the Type of Difference: The difference operation is chosen in order to remove the effect of the key used in the system under attack. Usually it is the xor difference which is the case in most cryptographic primitives but there are some primitives where the difference is addition or multiplication modulo n such as in IDEA and SAFER. For the sake of simplicity, suppose we have the following ciphertexts, $c_1 = m_1 \oplus k$ and $c_1 = m_2 \oplus k$. Then the suitable difference operation that cancels the key effect is the following: $c_1 \oplus c_2 = m_1 \oplus m_2$ which allows the attacker to gain some information (their difference) about the plaintexts. Another example, if $c_1 = m_1 \boxplus k$ and $c_2 = m_2 \boxplus k$ where $\boxplus$ is the addition modulo an integer, usually in most cryptographic primitives this is an 8-bit, 16-bit or 32-bit integer. Then the suitable difference that cancels the key effect is the following: $c_1 \boxplus c_2^{-1} = (m_1 \boxplus k) \boxplus (m_2 \boxplus k)^{-1} = m_1 \boxplus k \boxplus k^{-1} \boxplus m_2^{-1} = m_1 \boxplus m_2^{-1}$, where the inverse operation is taken under the group defined by the set of the possible plaintexts and under the $\boxplus$ operation. In fact nowadays the $\oplus$ is the most used operation in mixing the key with data as the the other operations such as addition makes the cipher difficult to analyze especially against differential cryptanalysis and thus designers tend to use $\oplus$ operations to ease the analysis and also because its cost is cheaper in hardware compared to the $\boxplus$.

Differential Characteristic and Differential: As most modern symmetric ciphers are iterated, i.e. they iterate for some rounds, where all rounds basically perform the same operations with the exception of using a specific round key or a round constant, linear and differential properties of such ciphers are analyzed for one round and then extended to multiple rounds. Differential attacks are based on the so-called *differential characteristic* which is a sequence of intermediate difference relations for all rounds where the probability of each element in this sequence determines the probability of the corresponding differential characteristic. The collection of all the r-round differential characteristics (aka trails or paths) with input $\alpha = \alpha_0$ and output $\beta = \alpha_r$ is called the *differential* of the difference approximation (α_0, α_r). Each r-round differential characteristic could be seen as a sequence $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$ where i defines the ith differential trail between α_0 to α_r.

On the Probability of a Differential Characteristic: As pointed in Chapter 1, iterated cryptographic primitives have both linear and nonlinear components. Clearly the linear operation preserves the difference and thus the output difference occurs with probability 1. However, the nonlinear components (usually the Sboxes or $\boxplus$ modulo an integer) do not preserve differences but some input differences can have some output differences with some probability. For instance if we have a nonlinear component $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where n and m are small integers then we can build its *difference approximation table*. This table simply contains the probabilities of all the possible input and output differences. Suppose we have input difference ΔX and output difference ΔY, then

$$S_{\Delta X,\Delta Y} = |\{X \in Domain(S) : S(X) \otimes (S(X \otimes (\Delta X)^{-1}))^{-1} = \Delta Y\}|$$

where $\otimes$ is the difference operation. The probability of having an input difference ΔX and getting an output difference ΔY is $\frac{S_{\Delta X,\Delta Y}}{2^n}$. This together with the concept of a Markov cipher which we define below enable us to compute the probability of a differential characteristic.

One class of iterated ciphers defined in [81], is called *Markov ciphers*. For such ciphers, under well established independence assumptions, the probability of a differential characteristic can be calculated. In these ciphers, the sequence $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$ is called a *Markov chain* if the intermediate transitions $\alpha_i \to \alpha_{i+1}$ are independent of each other, in other words, given the present state, the future and past states are independent. In other words,

$$\Pr((v_{i+1} = \alpha_{i+1})|v_i = \alpha_i, \cdots, v_0 = \alpha_0) = \Pr((v_{i+1} = \alpha_{i+1})|v_i = \alpha_i) \quad \text{for} \quad 0 \leq i \leq r$$

Moreover, if $\Pr(v_{i+1} = \beta|v_i = \alpha)$ is independent of i for all α and β then a Markov chain is called homogeneous or stationary, in other words $\Pr(v_{i+1} = \beta|v_i = \alpha) = \Pr(v_i = \beta|v_{i-1} = \alpha)$.

Definition 2. *An iterated cipher with round function $Y = f(X, K_i)$ is a Markov cipher if there is a difference operation where $\Pr(\Delta Y = \beta|\Delta X = \alpha, X = m)$ is independent of m for all α and β when the round key K_i is chosen uniformly at random.*

The probability of a differential relation can be computed using a difference transition matrix. In the following, we briefly describe how to construct and use these matrices.

Difference Transition Matrix [81]: Assume that we have an r-round Markov cipher with independent and uniformly random round keys. To estimate the probability of an r-round differential (α_0, α_r), we first consider the probability of each differential characteristic between α_0 and α_r, so the probability of the i-th differential characteristic $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$ is

$$p_i = \prod_{i=1}^{r} \Pr(F_K(X) \oplus F_K(X') = \alpha_i | X \oplus X' = \alpha_{i-1})$$

Consequently the probability of an r-round differential (α_0, α_r) is the sum of all the probabilities of all the possible differential characteristics between (α_0, α_r), that is $\sum_{i=1}^{N_d} p_i$, where N_d is the number of all the possible differential characteristics between (α_0, α_r). Let D denote the transition difference-probability matrix of an n-bit Markov cipher. D has size $(2^n - 1) \times (2^n - 1)$, the (i, j) entry in D is the probability of obtaining an output difference say β_j when we have an input difference say β_i, i.e. $\Pr(\Delta(F_K(X)) = \beta_j | \Delta(X) = \beta_i)$ where F_K is the round function of the Markov cipher. Now, for any r, the (i, j) entry of the matrix D^r, $p_{ij}^{(r)}$ is equivalent to the probability of the r-round differential (β_i, β_j).

Key Recovery and Data Complexity of a Differential Attack: Suppose we have an r-round iterated n-bit block cipher with k-bit secret key. To mount a differential attack we first look at a good $(r-1)$-round characteristic, i.e. $(\alpha_0, \cdots, \alpha_{r-1})$. In other words a characteristic occurring with a probability p significantly larger than 2^{-n}.[1]

Suppose we want to recover some key bits of the last round key k_r. We guess part of k_r that would enable us to partially decrypt a number of ciphertext pairs c_r^1 and c_r^2 obtained by encrypting a number of plaintext pairs m_1 and m_2 with difference $\Delta m = \alpha_0$. The guess and the partial decryption process should have a time complexity below the brute force complexity 2^k. Then for each key guess and each partial decryption of a given ciphertext pair, we check if the difference of the partially decrypted pair matches the characteristic value or part of it at round $r-1$, α_{r-1}. Then the key guess that yields the highest number of matches would probably be the right key. As noted in [81], when mounting a key recovery differential attack, we assume that all the round subkeys

[1] It was shown in [59] that the maximum differential probability in an n-bit random permutation is expected to be $\leq \frac{n}{2^{n-1}}$ with a higher probability that is almost equal to 1. Therefore the maximum differential approximation for a 64-bit and 128-bit block ciphers is expected to be less than 2^{-57} and 2^{-120} respectively but as pointed in [11] they are hard to find.

are independent and uniformly random to compute the differential probability but we are also assuming that all the secret keys behave similarly, i.e. for an $(r-1)$-round differential, $(\alpha_0, \cdots, \alpha_{r-1})$:

$$\Pr(\Delta c_{r-1} = \alpha_{r-1} | \Delta m = \alpha_0) \approx \Pr(\Delta c_{r-1} = \alpha_{r-1} | \Delta m = \alpha_0, k_1 = s_1, \cdots, k_{r-1} = s_{r-1})$$

for almost all subkey values $(s_1, \cdots, s_{r-1})$. This is known in the literature as the *hypothesis of stochastic equivalence* [81] (See Chapter 4). A successful differential attack requires data complexity equivalent to $\frac{c}{p}$, where c is a small integer since this will give us on average c *right pairs* which are pairs that follow the differential characteristic.

However, for the attack to succeed, the right value of k_r have to be suggested at a rate that distinguishes it from the other candidate values. Usually this is determined by the *signal-to-noise* concept which is often denoted as S/N and is defined as the number of times the right key guess is counted over the number of times a wrong key guess is counted. For more details we refer the reader to [14, 79].

The work of Selçuk in [104] reveals that the targeted key length is one of the factors that affects the success probability of a differential attack besides the S/N rate and the available amount of plaintext. Approximating the binomial distribution of the key counters to a normal distribution, Selçuk provided a formula depending on the key length and the amount of plaintext pairs to estimate the success probability of a differential attack.

However, the recent work of Blondeau and Gérard [18] provides a better statistical framework to compute the success probability and data complexity in differential cryptanalysis as their statistical analysis deals with the binomial distribution of the counters which is believed to be better than approximating the binomial distribution to a normal distribution [18].

Several extensions to the normal differential attack have been developed in the 90's. However, unlike the normal differential their applicability depends on the cryptographic primitive under attack. Below we discuss some of them.

Truncated Differential Cryptanalysis: In their original differential attack [13], Biham and Shamir considered the idea of using many differentials with different input difference but the same output difference. Later Knudsen in [76] used the reverse of this idea. So rather than looking into multi inputs and one output difference, he looked at

one input but multi output differences or part of the output difference. This technique is called *truncated differential* as in the multi output differences we only consider part of the bits which are the ones having the same value and position. This will definitely give us a higher probability than considering only one output difference. However unlike differential cryptanalysis, truncated differentials are limited and it might not be applicable on some cryptographic primitives designs but when applicable they are more effective than the usual differential attacks as they would result in either attacking more rounds or reducing the data complexity required to have a successful attack. More recently in [19], Blondeau and Gérard combined the two above ideas, i.e. multi input differentials and multi output differences, and proposed *multiple differential cryptanalysis* along with a statistical framework to estimate its data complexity, time complexity and success probability.

Higher Order Differential Cryptanalysis: This technique was introduced by Lai in [80] and used by Knudsen in [76] to develop an attack against ciphers presumably secure against ordinary differential attacks. Instead of considering a difference between two values as in differential attacks, higher order differential attacks consider difference between differences. For instance, let $f : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$ be a Boolean function. Consider a normal α_1 difference $f(x) \oplus f(x \oplus \alpha_1)$, this is called a 1st order difference (or derivative), an α_2 difference of $\Delta_{\alpha_2} = f(x) \oplus f(x \oplus \alpha_1)$ is:

$$\Delta_{\alpha_2}(\Delta_{\alpha_1} f(x)) = f(x) \oplus f(x \oplus \alpha_1) \oplus f(x \oplus \alpha_2) \oplus f(x \oplus \alpha_1 \oplus \alpha_2)$$

More generally a dth-order differential is defined as $\Delta_{\alpha_d}(\Delta_{\alpha_1,\cdots,\alpha_{d-1}} f(x))$. Since any ciphertext bit can be expressed as a polynomial in the secret key bits and plaintext bits, ciphers with low algebraic degree are weak against this technique as they could be attacked using the fact that the value of the $(d+1)$-order derivative is zero if $f(x)$ has degree d. For instance, if we know that the algebraic degree of a cryptographic primitive is 16, then the key guess that makes the value of the 17th order differential zero will be the right key.

One application of higher order differentials is called *cube attacks*. They were introduced by Dinur and Shamir in [47]. Cube attacks use higher order differentials in order to obtain linear equations in the secret key bits of the target cipher even if the attacker has no idea about the design of the target cipher, i.e. the multivariate polynomials that represent each bit at the output. Like higher order differentials attacks, if there

is a single bit represented by a polynomial whose Algebraic Normal Form (ANF) has a low degree then cube attacks can be applied successfully. Thus, as block ciphers have high algebraic degree even for a small number of rounds, cube attacks can only be effective for stream ciphers as they use nonlinear feedback registers which provide a slow diffusion and consequently some output bits might have low algebraic degree even after a high number of rounds. The attack was successfully applied to reduced-rounds of Trivium [32].

More recently, an improvement to cube attacks called *dynamic cube attacks* has been proposed [48]. The attack tries to benefit from high algebraic degree polynomials P that can be decomposed into three polynomials P_1, P_2 and P_3 such that $P = P_1P_2 + P_3$ where P_1 is called the source polynomial, P_2 is called the target polynomial and it is very dense and has a high algebraic degree and P_3 is called the remainder polynomial and it has a low algebraic degree. The idea of the attack is to choose some inputs called *dynamic variables* that nullify the polynomial P_1 and therefore simplify the polynomial P to $P = P_3$. The choice of P_1 and P_2 requires a careful analysis of the cipher under consideration. The attack was successfully applied to the full Grain-128 but only for some keys belonging to a subset of 2^{-10} of all the possible 2^{128} keys.

Impossible Differential Cryptanalysis: Rather than exploiting a differential with a high probability, impossible differential attacks exploit differentials with a very low or zero probability. The use of a zero differential probability was firstly used by Knudsen in his analysis of his block cipher DEAL [74]. Later the use of a zero or low differential probabilities was exploited by Biham et al in their analysis of SKIPJACK [12] where they named it *impossible differential*.

2.2 Linear Cryptanalysis

Linear Cryptanalysis is a known plaintext attack that exploits ciphers which are close to linear functions. The attack was invented by Matsui to break the DES cipher in [86]. The idea is to find a linear relation between some plaintext bits and ciphertext bits (and also some secret key bits in the case of block ciphers) and then exploit the bias or the correlation of this linear relation which should be unbiased (uncorrelated) in an ideal cipher.

Suppose we have an n-bit plaintext and ciphertext denoted by m and c respectively and obtained under a secret key k of length l. Then linear cryptanalysis finds an α and $\beta \in \mathbb{F}_2^n$ such that probability of linear approximation

$$\langle \alpha, m \rangle \oplus \langle \beta, c \rangle = \langle \gamma, k \rangle, \quad \text{where} \quad k \in \mathbb{F}_2^l$$

deviates significantly from $\frac{1}{2}$.

We start by introducing the Walsh transform of a single and vectorial Boolean functions respectively. We then define a linear approximation in terms of the Walsh transform and this is useful especially when we discuss the multidimensional linear attacks.

The Walsh transform of a Boolean function $f(x)$ where $f : \mathbb{F}_2^n \to \mathbb{F}_2$ at the point $a \in \mathbb{F}_2^n$ is defined to be the real valued function

$$\hat{f}(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus \langle a, x \rangle}.$$

Similarly the Walsh transform (or the Fourier coefficient) of a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ at the point $(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ is defined by

$$\hat{F}(a, b) = \sum_{x} (-1)^{\langle b, F(x) \rangle \oplus \langle a, x \rangle}$$

In linear cryptanalysis we approximate the function $f(x)$ to $\langle a, x \rangle$. Therefore, we are interested in computing $Pr(f(x) = \langle a, x \rangle)$. Let N_0 denote the number of $x = u$ such that $f(u) = \langle a, u \rangle$ and let N_1 denote the number of $x = u$ such that $f(u) = \langle a, u \rangle \oplus 1$. Obviously, $N_0 + N_1 = 2^n$ and the Walsh transform can be written as follows

$$\hat{f}(a) = N_0 - N_1$$

Let $p_a = Pr(f(x) = \langle a, x \rangle)$. So $p_a = \frac{N_0}{2^n}$ and we see that $\hat{f}(a) = 2N_0 - 2^n = 2^{n+1}(p_a - \frac{1}{2})$. Consequently

$$p_a = \frac{1}{2} + \frac{1}{2^{n+1}} \hat{f}(a)$$

Similarly for a vectorial function $F(x)$, the probability that $\langle b, F(x) \rangle$ equals $\langle a, x \rangle$ can

be calculated as follows

$$p_{a,b} = \frac{1}{2} + \frac{1}{2^{n+1}}\hat{F}(a,b)$$

The bias of approximating $\langle b, F(x)\rangle$ by $\langle a, x\rangle$ is defined as $\epsilon = p_{a,b} - \frac{1}{2} = \frac{\hat{F}(a,b)}{2^{n+1}}$ and the correlation of of approximating $\langle b, F(x)\rangle$ by $\langle a, x\rangle$ is defined as

$$C(a,b) = 2\epsilon = \frac{\hat{F}(a,b)}{2^n} \tag{2.1}$$

Linear Characteristic and Linear Hull Linear attacks are based on the so-called *linear characteristic* which is a sequence of intermediate linear relations for all rounds where the correlation of each element in this sequence determines the correlation of the corresponding linear characteristic. The collection of all the linear characteristics with input mask $\alpha = \alpha_0$ and output mask $\beta = \alpha_r$ is often called the *linear hull* of the linear approximation (α_0, α_r), also each r-round linear characteristic could be seen as a sequence $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$ where i defines the i-th linear trail between α_0 to α_r.

On the Correlation of a Linear Characteristic: As pointed in Chapter 1, iterated cryptographic primitives have both linear and nonlinear components. Clearly the linear operation preserves a linear approximation with probability 1. However, the nonlinear components (usually the Sboxes or $\boxplus$ modulo an integer) need to be linearly approximated with some probability. For instance if we have a nonlinear component $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ where n and m are small integers then we can build its *linear approximation table*. This table simply contains the correlations of all the possible input and output approximations (aka input and output masks).

Suppose we have input mask α and output mask β, then

$$S_{\alpha,\beta} = |\{X \in Domain(S) : \langle \beta, S(X)\rangle = \langle X, \alpha\rangle\}|$$

The probability of an input mask α being equal to an output mask β is

$$\Pr(\langle \beta, S(x)\rangle = \langle \alpha, x\rangle) = \frac{S_{\alpha,\beta}}{2^n}$$

and their correlation is equivalent to $2\Pr(\langle \beta, S(x)\rangle = \langle \alpha, x\rangle) - 1$.

The Piling-Up Lemma [86] was used by Matsui to estimate the probability of a linear

characteristic but the estimation of the correlation of a linear hull results from the theory of correlation matrices [41, 44] or the Walsh tansform of composite mappings [35].

Correlation Matrix [41, 44]: Assume that we have a composite function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $F = F_r \circ \cdots \circ F_2 \circ F_1$. To estimate the correlation of an r-round linear approximation (α_0, α_r) of F, we first consider the correlation of each linear characteristic between α_0 and α_r. Using the Piling-Up Lemma and assuming that all the intermediate linear approximations, i.e. $\langle \alpha_{(j-1)i}, c_{j-1} \rangle \oplus \langle \alpha_{ji}, c_j \rangle = \langle \gamma_j, k_j \rangle$, are independent and $\langle \gamma_j, k_j \rangle = 0$ with probability p_j for all j's, one can see that the probability of the ith linear characteristic $(\alpha_0 = \alpha_{0i}, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r = \alpha_{ri})$ is

$$\frac{1}{2} + 2^{r-1} \prod_{j=1}^{r} \left(p_j - \frac{1}{2} \right)$$

Consequently, the correlation of ith linear characteristic $(\alpha_0 = \alpha_{0i}, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r = \alpha_{ri})$ is

$$C_i = \prod_{j=1}^{r} C_{F_j}(\alpha_{(j-1)i}, \alpha_{ji}).$$

It is well known, see e.g., [44], that the correlation of a linear approximation is the sum of all correlations of linear trails starting with the same mask α and ending with the same mask β, i.e., $C_F(\alpha_0, \alpha_r) = \sum_{i=1}^{N_l} C_i$, where N_l is the number of all the possible linear characteristics between (α_0, α_r).

The correlation of a linear trail is key dependent but only the sign of the correlation depends on the key. More precisely, in [44] the following formulas were proven under the assumption that we have a key-alternating cipher[2]:

$$C_i = (-1)^{s_i} \prod_{j=1}^{r} C_{F_j}(\alpha_{(j-1)i}, \alpha_{ji}),$$

where $s_i \in \mathbb{F}_2$ depends on the i-th linear characteristic and the round keys.

Therefore, the correlation of the linear hull (α, β) is

$$C_F(\alpha, \beta) = \sum_{i=1 | \alpha_i = (\alpha = \alpha_{0i}, \cdots, \alpha_{(r-1)i}, \beta = \alpha_{ri})}^{N_l} (-1)^{s_i \oplus d_i} |C_i| \tag{2.2}$$

[2]Key-alternating ciphers are a subclass of Markov ciphers that alternate key additions with key-independent rounds.

where the sign $d_i \in \mathbb{F}_2$ denotes the sign of the correlation, C_i.

Let C denote the correlation matrix of an n-bit key-alternating cipher. C has size $2^n \times 2^n$ (or $(2^n-1)\times(2^n-1)$ when we exclude the zero input or output masks), the (i,j) entry in C corresponds to the correlation of an input mask, say β_i, and output mask, say β_j, i.e. $C_F(\beta_i, \beta_j) = 2\Pr(\langle \beta_i, x\rangle = \langle \beta_j, F(x)\rangle) - 1$, where F is the un-keyed composite function of the key-alternating cipher and 'x' is its input. Now the correlation matrix for the keyed round function is obtained by changing the signs of each row in C according to the round subkey bits or the round constant bits involved.

Key Recovery and Data Complexity of a Linear Attack: A basic linear attack recovers one bit of information about the secret key. This is often called in the literature as Matsui's algorithm 1 which is a distinguishing attack of the cipher under consideration since it doesn't recover the whole key but distinguishes the cipher from a randomly chosen permutation. The following algorithm describes the basic linear attack.

Algorithm 1 Matusi's distinguishing attack

Require: N known plaintexts (m, c)
Require: $T_b \equiv$ the number of times $\langle m, \alpha\rangle \oplus \langle c, \beta\rangle = b$ where $b \in \mathbb{F}_2$
1: **for** all the N known plaintexts (m, c) **do**
2: Compute $b = \langle m, \alpha\rangle \oplus \langle c, \beta\rangle$
3: Increment T_b
4: **end for**
5: **if** $T_0 > \frac{N}{2}$ **then**
6: Guess $\langle \gamma, k\rangle = 0$.
7: **else**
8: Guess $\langle \gamma, k\rangle = 1$.
9: **end if**

In the following we estimate the data complexity needed to perform the basic linear attack successfully. Without loss of generality, assume that $T_0 > \frac{N}{2}$ and that $m\cdot\alpha \oplus c\cdot\beta = 0$ holds with probability $p = \frac{1}{2} + \epsilon$ where $\epsilon > 0$. Then

$$\Pr(\text{Success}) = \Pr\left(T_0 > \frac{N}{2}\right) = 1 - \Pr\left(T_0 \leq \frac{N}{2}\right)$$

T_0 is random variable and it is the sum of N identically distributed and mutually independent binary random variables $X_i \in F_2$ which have the following probability

distribution.

$$\Pr(X = X_i) = \begin{cases} \frac{1}{2} + \epsilon, & X_i = 1 \\ \frac{1}{2} - \epsilon, & X_i = 0 \end{cases}$$

Clearly T_0 follows a binomial distribution with an expected value $E(T_0) = Np = N(\frac{1}{2}+\epsilon)$ and variance $V(T_0) = Np(1-p) = N(\frac{1}{4} - \epsilon^2)$. So $T_0 \sim B(N(\frac{1}{2} + \epsilon), N(\frac{1}{4} - \epsilon^2))$. When N is large, then using the Central Limit Theorem the probability distribution of

$$x = \frac{T_0 - Np}{\sqrt{Np(1-p)}} = \frac{T_0 - N(\frac{1}{2} + \epsilon)}{\sqrt{N(\frac{1}{4} - \epsilon^2)}}$$

can be approximated by the standard normal distribution $\Phi(x)$, i.e. $x \sim \mathcal{N}(0, 1)$. Therefore,

$$\Pr(\text{Success}) = 1 - \Pr\left(T_0 - \frac{N}{2} - \epsilon N \leq -\epsilon N\right) = 1 - \Pr\left(\frac{T_0 - \frac{N}{2} - \epsilon N}{\sqrt{N(\frac{1}{4} - \epsilon^2)}} \leq \frac{-\epsilon N}{\sqrt{N(\frac{1}{4} - \epsilon^2)}}\right)$$

Assume that ϵ is small, then ϵ^2 can be ignored, Thus

$$\Pr(Success) = 1 - \Pr\left(x \leq \frac{-\epsilon N}{\sqrt{N(\frac{1}{4} - \epsilon^2)}}\right) \approx 1 - \Phi(-2\epsilon\sqrt{N})$$

When $N = \epsilon^2$,

$$\Pr(Success) = 1 - \Phi(-2) = 1 - 0.02 = 0.98$$

Therefore having data complexity $N = \frac{c}{\epsilon^2}$ where c is a small integer, the success rate will be almost 1. More key information can be recovered using a simple algorithm similar to the key recovery procedure in the differential attack. Suppose we have an r round cipher with a round function f, we use an $(r-1)$ linear approximation to recover $\log_2(l)$ bits from the last round key k_r by the following algorithm which is known in the literature as Matsui's algorithm 2.

Unlike the distinguishing attack whose success rate is easily evaluated as discussed above, the success rate of the key recovery attack is more complicated. One assump-

Algorithm 2 Matusi's algorithm 2

Require: N known plaintexts (m, c)
Require: $(r-1)$ linear approximation (α_0, α_{r-1})
Require: $2l$ key counters, namely $T_{0,0} \cdots, T_{0,l-1}$ and $T_{1,0} \cdots, T_{1,l-1}$
1: **for** all the N known plaintexts (m, c) **do**
2: **for** all the l guessed key values for k_r **do**
3: Compute $b = m \cdot \alpha_0 \oplus f^{-1}(c, l) \cdot \alpha_{r-1}$
4: Increment $T_{b,l}$
5: **end for**
6: **end for**
7: Find the counter $T_{i,t}$ that have the maximum value
8: Guess that $k_r = t$

tion that is used in determining the success rate is that wrong key guesses to the key information results in having random values for $\hat{c}_{r-1} = f^{-1}(c, l)$. In other words $\hat{c}_{r-1}$ is treated as being equivalent to the ciphertext obtained after $(r+1)$ rounds, c_{r+1}. This is known in the literature as the *hypothesis of wrong key randomization* [58]. For more details about the success rate, we refer the reader to the work of [104] as it provides a formula depending on the key length and the amount of plaintext to estimate the success probability of a linear attack. As noted in [104], approximating the binomial distribution of the counters to a normal distribution gives accurate results in linear attacks compared to differential attacks.

Several papers tried to extend linear cryptanalysis by using multiple linear approximations [10, 15, 62, 63, 70] to either reduce the data complexity or gain more information about the key. Below we discuss the multidimensional linear attack [62, 63] which is an improvement over all the previously proposed attacks that use multiple linear approximations.

Multidimensional Linear Attack: We start by defining the mutual capacity between two probability distributions and the squared Euclidean distance and show its relation to the capacity since they are both used in estimating the data complexity of the multidimensional linear attacks.

The term capacity was firstly defined and used by Biryukov et al. in [15] where they estimated the data complexity to be inversely proportional to the capacity of the multiple linear approximations. Later the term was generalized by Hermelin et al. in [62] in order to estimate the data complexity of their multidimensional linear attack.

Definition 3. *Let* $p = (p_0, \cdots, p_M)$ *and* $q = (q_0, \cdots, q_M)$ *be two probability distributions. Their mutual* capacity *is then*

$$\mathrm{Cap}(p, q) = \sum_{i=0}^{M} \frac{(p_i - q_i)^2}{q_i}$$

Thus given a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the mutual capacity of the probability distribution p of the output of F and the uniform distribution is called the capacity of p and is given by

$$\mathrm{Cap}(p) = \sum_{y \in \mathbb{F}_2^m} \frac{(2^{-n} \cdot |\{x \in \mathbb{F}_2^n \mid F(x) = y\}| - 2^{-m})^2}{2^{-m}} \tag{2.3}$$

The *Squared Euclidean distance* is defined as follows

$$\mathrm{D}(p) = \sum_{y \in \mathbb{F}_2^m} \left(2^{-n} \cdot |\{x \in \mathbb{F}_2^n \mid F(x) = y\}| - 2^{-m}\right)^2 \tag{2.4}$$

and it differs from the capacity by a factor of 2^m, i.e. $D(p) = 2^{-m} \mathrm{Cap}(p)$.

Suppose that we have a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$. Let $\alpha_i \in \mathbb{F}_2^n$ and $\beta_i \in \mathbb{F}_2^n$ be input and output masks respectively. Define the following g_i functions where $1 \leq i \leq m$:

$$g_i(x) = \langle \beta_i, F(x) \rangle \oplus \langle \alpha_i, x \rangle$$

Denote by C_i the correlation of g_i, $1 \leq i \leq m$ and let them be defined as the base correlations. Let $C(0, a)$ refers to the correlation of the linear approximation $\langle a, g_i(x) \rangle$. For instance if $a = e_i = (0, 0, \cdots, 1, \cdots)$ is a unit vector with 1 at position i, then $C(0, a) = C(0, e_i) = C(\alpha_i, \beta_i)$ and this holds for $1 \leq i \leq m$. When $a \neq e_i$, then $C(0, a)$ corresponds to a correlation of one of the combined approximations, i.e. if $a = e_i \oplus e_j$ then $C(0, a)$ is the correlation of the linear approximation $g_i(x) \oplus g_j(x) = \langle \beta_i + \beta_j, F(x) \rangle \oplus \langle \alpha_i + \alpha_j, x \rangle$. In other words $C(0, a) = C(\alpha_i \oplus \alpha_j, \beta_i \oplus \beta_j)$.

Let $p = (p_0, p_1, \cdots, p_{2^m - 1})$ be the probability distribution of the above m-dimensional linear approximations. In order to estimate the capacity of the distribution in terms of the correlations of the corresponding linear approximations we use the following Lemma which was proved in [62].

Lemma 2.2.1. *Let $g : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function with probability distribution p and single linear correlations $C(0,a)$ of $\langle a, g\rangle$. Then*

$$p_i = 2^{-m} \sum_{a \in \mathbb{F}_2^m} (-1)^{\langle a,i\rangle} C(0,a).$$

It is noted in [62] that the capacity can be expressed in terms of correlations or Fourier coefficients of the linear approximations involved in the multidimensional distribution by using the above Lemma in addition to the following relation which is known as Parseval's relation [34]

$$\sum_{u \in \mathbb{F}_2^n} \hat{\varphi}(u)^2 = 2^n \sum_{x \in \mathbb{F}_2^n} \varphi(x)^2,$$

where $\hat{\varphi}(u) = \sum_{x \in F_2^n} \varphi(x)(-1)^{u \cdot x}$ and $\varphi(x) \in \{1, -1\}$ for any $x \in \mathbb{F}_2^n$. In the following proposition, we show these relations between the capacity and the correlations or Fourier coefficients.

Proposition 2.2.2. *[62] Let $g = (g_1, \cdots, g_m)$ be a an m-dimensional vectorial Boolean function, where g_i is the boolean function defined above. Suppose that g has a probability distribution $p = (p_0, \cdots, p_{2^m-1})$. Then*

$$\mathrm{Cap}(p) = \sum_{a \neq 0} (C(0,a))^2 = 2^{-2n} \sum_{a \neq 0} (\hat{F}(0,a))^2.$$

Proof. From the above lemma, we know that

$$p_i = 2^{-m} \sum_{a \in F_2^m} (-1)^{\langle a,i\rangle} C(0,a)$$

where $1 \leq i \leq 2^m$. Now using Parseval's relation, we see that

$$2^m \sum_i p_i^2 = \sum_a C(0,a)^2.$$

Therefore,

$$\begin{aligned}
\text{Cap}(p) &= 2^m \sum_i (p_i - 2^{-m})^2 = 2^m \sum_i p_i^2 - 2^{-m+1} p_i + 2^{-2m} \\
&= 2^m \sum_i p_i^2 - 2^m \sum_i 2^{-m+1} p_i + 2^m \sum_i 2^{-2m} \\
&= \sum_a C(0,a)^2 - 2 \sum_i p_i + 1 = \sum_a C(0,a)^2 - 1 \\
&= \sum_{a \neq 0} (C(0,a))^2 = 2^{-2n} \sum_{a \neq 0} (\hat{F}(0,a))^2
\end{aligned}$$

□

In [63], two different statistical methods were proposed to recover the secret key using m-dimensional linear approximations. Namely, the goodness of fit method and this is usually based on χ^2-statistic and the method of distinguishing an unknown probability distribution from a given set of probability distributions and this is usually based on Log Likelihood Ratio (LLR-statistic). Below we only describe the χ^2 method which is the one that was applied to the block cipher PRESENT. For more details about the LLR method, we refer the reader to [63]. It has been noted in [38] that the LLR method cannot be applied in PRESENT as the probability distribution of the linear approximations in PRESENT depends heavily on the key (See Chapter 3 and Chapter 4).

Suppose we would like to recover l bits of the target key. Then for each key value $k \in \mathbb{F}_{2^l}$, we obtain the empirical probability distribution for each key $q_k = \{q_{k,0}, q_{k,1}, \cdots, q_{k,2^m-1}\}$ by computing the probability of the m-dimensional vectors which are Boolean values of m linear independent approximations. Now given the empirical probability distribution q_k, a key recovery attack is mounted by computing the χ^2 statistic

$$S(k) = 2^m \sum_{i=0}^{2^m-1} \left(q_{k,i} - 2^{-m}\right)^2$$

which also represents the squared Euclidean distance from the uniform distribution. Now according to the wrong key hypothesis mentioned above, the right key value is the one that yields the maximum squared Euclidean distance from the uniform distribution. Therefore, the right key is the one for which the statistic $S(k)$ is the largest. For more details about the success rate and data complexity of the multidimensional attack using the χ^2 method, we refer the reader to [63].

Recently a new attack called the statistical saturation attack [39] was shown in [82] to be similar to multidimensional linear attacks. Below we describe the statistical saturation attack and show its link to the multidimensional linear attack.

Statistical Saturation Attack: The statistical saturation attack was developed in [39] by Collard and Standaert to attack reduced-round PRESENT. The attack fixes some plaintext bits and exploits the bias at some output bits. The attack computes the Squared Euclidean distance between the distribution of the targeted output bits and the uniform distribution. The attack breaks 24 rounds of PRESENT. The data complexity of their attack was estimated to be proportional to the inverse of the squared Euclidean distance. To estimate the Squared Euclidean distance the authors propose an algorithm to calculate the theoretical distribution of 8 output bits of PRESENT. The algorithm guesses some subkey bits at each round and therefore cannot be applied for several number of rounds. Later, Leander [82] provided a useful method to estimate the average capacity of statistical saturation attacks. His method shows that statistical saturation attacks are closely related to multi-dimensional linear attacks. In the following we discuss Leander's method.

Let $e : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an encryption function. Now write e as follows $e : \mathbb{F}_2^r \times \mathbb{F}_2^s \to \mathbb{F}_2^t \times \mathbb{F}_2^u$ where

$$e(x, y) = \left(e^{(1)}(x, y), e^{(2)}(x, y)\right) \tag{2.5}$$

where $r + s = t + u = n$, $e^{(1)}(x, y) \in \mathbb{F}_2^t$ and $e^{(2)}(x, y) \in \mathbb{F}_2^u$.

The statistical saturation attack fixes 's' bits and considers 't' bits at the output. For simplicity we fix the last 's' bits and consider the distribution of the first 't' bits. Denote this restriction of e by $h_y : \mathbb{F}_2^r \to \mathbb{F}_2^t$ where

$$h_y(x) = e^{(1)}(x, y) \tag{2.6}$$

where $y \in \mathbb{F}_{2^s}$ denotes the fixed 's' input bits, and let p_y be the probability distribution of the output 't' bits of h_y. Leander provided a formula to estimate the average capacity of

p_y. The formula was proven using the following proposition which establishes a relation between the Fourier of a Boolean function and the Fourier of its restrictions.

Proposition 2.2.3. *[33] Let E and E' be subspaces of $\mathbb{F}_{2^n}$ such that $E \cap E' = \{0\}$ and whose direct sum equals $\mathbb{F}_{2^n}$. For every $a \in E'$, let h_a be the restriction of a Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ to the coset $a + E$ (h_a can be identified with a function on $\mathbb{F}_{2^k}$ where k is the dimension of E). Then*

$$\sum_{u \in E^\perp} (\hat{f}(u))^2 = |E^\perp| \sum_{a \in E'} (\hat{h}_a(0))^2$$

The following theorem gives the formula.

Theorem 2.2.4. *[82] With the above described notation, the average capacity of statistical saturation attacks where the average is taken over all possible fixations $y \in \mathbb{F}_{2^s}$ is given by*

$$\overline{\mathrm{Cap}(p_y)} = 2^{-2n} \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} (\hat{e}(a,b))^2 = \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} (C(a,b))^2 \tag{2.7}$$

where $C(a,b)$ refers to the correlation of approximating $\langle b, e(x_1, x_2)\rangle$ by $\langle a, (x_1, x_2)\rangle$ where $x_1 \in \mathbb{F}_2^r$ and $x_2 \in \mathbb{F}_2^s$.

Proof. [82]

The average capacity of p_y over all the possible y fixations is given by

$$\overline{\mathrm{Cap}(p_y)} = 2^{-s} \sum_{y \in \mathbb{F}_{2^s}} \mathrm{Cap}(p_y)$$

Now since h_y is defined as follows $h_y : \mathbb{F}_2^r \to \mathbb{F}_2^t$, then using proposition 2.2.2, we see that

$$\overline{\mathrm{Cap}(p_y)} = 2^{-s} \sum_{y \in \mathbb{F}_2^s} \mathrm{Cap}(p_y) = 2^{-s} \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} 2^{-2r} (\hat{h}_a(0,b))^2$$

Now applying Proposition (2.2.3) on the function $\langle b, e\rangle$ and its restriction $\langle b, h_y\rangle$ where we choose $E = \mathbb{F}_2^r \times \{0\}$ and $E' = E^\perp = \{0\} \times \mathbb{F}_2^s$ gives us

$$\sum_{u \in \{0\} \times \mathbb{F}_2^s} (\hat{e}(u,b))^2 = 2^s \sum_{a \in \{0\} \times \mathbb{F}_2^s} (\hat{h_a}(0,b))^2$$

Now using the above equations, we see that

$$\overline{\mathrm{Cap}(p_y)} = 2^{-s} \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} 2^{-2r} (\hat{h}_a(0,b))^2 = 2^{-2s-2r} \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} (\hat{e}(a,b))^2$$

Now applying Identity (2.1) to the above equation, we see that

$$\overline{\mathrm{Cap}(p_y)} = 2^{-2n} \sum_{a \in \{0\} \times \mathbb{F}_2^s, b \in \mathbb{F}_2^t \times \{0\}} (\hat{e}(a,b))^2 = (C(a,b))^2$$

□

Clearly the last equality shows that statistical saturation attacks and multidimensional attacks are very similar attacks since the average capacity of the statistical saturation attack and the capacity of multidimensional linear approximations (shown in proposition 2.2.2) are equivalent.

Statistical saturation attacks perform well when we identify any input subspace $U \subset \mathbb{F}_{2^n}$ and any output subspace $V \subset \mathbb{F}_{2^n}$ that make the sum $\sum_{u \in U, v \in V} (C_F(u,v))^2$ big (U corresponds to the input masks and V corresponds to the output masks).

The average squared correlation (aka potential linear approximation [90] or expected linear probability [45], see 4.2.2) of the linear approximation (a,b) is used to estimate the average data complexity of multidimensional attacks or statistical saturation attacks over all the key space. Specifically, the data complexity of statistical saturation attacks is estimated to be proportional to the inverse of $\mathrm{D}(p_y) = 2^{-t}\overline{\mathrm{Cap}(p_y)}$ according to the original attack paper [39] and to also some experiments in [82].

While Theorem 2.2.4 considers an input subspace consisting of fixing the last s input bits and an output subspace consisting of the first t output bits, the Theorem also holds in general as noted in [82]. For instance, we can restrict the above defined encryption function e to the case where one fixes the first r bits in the input to y and considers only the first t bits of the output. Thus e can be written as follows $e : \mathbb{F}_2^r \times \mathbb{F}_2^s \to \mathbb{F}_2^t \times \mathbb{F}_2^u$ where

$$e(y,x) = \left(e^{(1)}(y,x), e^{(2)}(y,x)\right) \tag{2.8}$$

and its restriction h_y can be redefined as follows $h_y : \mathbb{F}_2^s \to \mathbb{F}_2^t$ where

$$h_y(x) = e^{(1)}(y, x) \tag{2.9}$$

Now following the proof of Theorem 2.2.4 and choosing $E = \{0\} \times \mathbb{F}_2^s$ and $E' = E^{\perp} = \mathbb{F}_2^r \times \{0\}$, one can see that

$$\overline{Cap(p_y)} = 2^{-2n} \sum_{a \in \mathbb{F}_2^r \times \{0\}, b \in \mathbb{F}_2^t \times \{0\}} (\hat{e}(a, b))^2 = \sum_{a \in \mathbb{F}_2^r \times \{0\}, b \in \mathbb{F}_2^t \times \{0\}} (C(a, b))^2 \tag{2.10}$$

In Chapter 6, we use equation 2.10 to show the existence of linear approximations with large correlations in PRINTCIPHER for some weak keys in the sense of the invariant subspace attack described in Chapter 6.

Zero Correlation Cryptanalysis: More recently, Bogdanov and Rijmen in [25] proposed a new attack called *Zero-correlation cryptanalysis* and as the name suggests the attack exploits linear approximations having a zero correlation for any key value. Thus, zero correlation attacks are the linear equivalent of impossible differential attacks. The authors propose an algorithm that computes the exact correlation of a linear approximation using 2^{n-1} chosen plaintexts, where n is the block size of the cipher. The algorithm is used to distinguish a block cipher with a key length larger than its block size from truly random ciphers which deviate from zero with a high probability. The authors showed examples of zero correlations linear approximations of reduced rounds of AES and other ciphers and they used them to mount a key recovery attack on reduced rounds of AES-192 and AES-256 as well as other ciphers.

Moreover, in [26], the authors propose a statistical technique to significantly reduce the data complexity using many zero correlation linear approximations and they applied their technique in reduced rounds of TEA and XTEA. For instance, when using l zero correlation approximations the data complexity will be in the order of $\frac{2^n}{\sqrt{l}}$ which is a significant reduction compared to the 2^{n-1} data complexity when only one zero correlation approximation is used. More recently, in [24], the authors propose a multidimensional zero correlation attack that uses multiple zero correlation approximations regardless of their statistical independence.

2.3 Estimating the Probabilities of Differential and Linear Approximations

There are two known automated methods for estimating the differential probability of a differential approximation and the correlation value of a linear approximation.

The first method uses a submatrix of the difference transition matrix or a submatrix of the correlation matrix as the difference transition and correlation matrices of an n-bit block cipher have size $(2^n-1)\times(2^n-1)$ which is huge for a practical block cipher. The difference transition submatrix was firstly proposed and used by Lai and Massey in their differential estimations of the block cipher PES which was improved to the famous block cipher IDEA [81]. In their differential analysis of PES, an 8×8 submatrix was used. This submatrix approach has also been used recently to estimate the differential probabilities of the 64-bit block cipher Maya [52] where a 64×64 submatrix was used [27]. The correlation submatrix approach has been recently used in estimating the correlation of the linear approximations in SPONGENT [21].

The second method uses a branch and bound algorithm. This method was firstly proposed and used by Matsui to find the best differential characteristic of the DES cipher [87]. As noted by Matsui, the algorithm can also be used to find the best linear characteristic. Matsui's branch and bound algorithm can be seen as a depth first search algorithm that first finds the best $(r-1)$-round differential or linear characteristic and then finds the best r-round differential or linear characteristic. The algorithm builds trees whose roots are input differences or masks and whose nodes at level i correspond to the i-round output differences or output masks. As the number of rounds increases, it becomes infeasible to keep track of all the possible output differences or masks. Therefore, the algorithm cuts the branches that lead to a probability that is lower than an initial specified r-round lower bound probability value. The following recursive procedures describe Matsui's branch and bound algorithm.

Several techniques could be used to reduce the search space of the possible ΔX_1 and ΔY_i in the above algorithm. For instance, we can restrict the input differences to activate as few Sboxes as possible since this usually results in obtaining higher probabilities. As Matusi's branch and bound finds only the best differential, in order to estimate the probability of a difference approximation we need to adjust the branch and bound algorithm to find the best 'm' differential characteristics (with the same input and output difference) probabilities and add all these 'm' probabilities to get an estimate of the total probability of the differential. Similarly, in order to estimate the correlation or the average squared correlation of a linear hull, we need to find the best 'm' linear characteristics (with the same input and output mask) and add them to get the correlation in the case that the key is known or the average squared correlation in the general case where the key is not known of a linear hull.

Algorithm 3 Matusi's branch and bound algorithm

Require: Initial probability values $\{B_1, B_2, \cdots, B_r\}$, where each B_i is the initial estimate of the best i-round differential characteristic probability.

Require: Define $(\Delta X_i, \Delta Y_i)$ as the probability of having an input difference ΔX_i at round i with an output difference ΔY_i at round i.

Ensure: $B_r \equiv$ Best r-round differential characteristic probability

BEGIN main procedure
for each possible candidate ΔX_1 **do**
  $p_1 = \max_{\Delta Y_1}(\Delta X_1, \Delta Y_1)$
  if $p_1 \times B_{r-1} \geq B_r$ **then**
    Call **procedure** Round-2
  end if
end for
EXIT main procedure

procedure Round-i $(2 \leq i \leq r-1)$
for each possible candidate ΔY_i **do**
  Let $\Delta X_i = \Delta Y_{i-1}$ and $p_i = (\Delta X_i, \Delta Y_i)$.
  if $p_1 \times p_2 \times \cdots \times p_i \times B_{r-i} \geq B_r$ **then**
    Call **procedure** Round-$(i+1)$
  end if
end for
Return to the upper **procedure**.

procedure Round-r
Let $\Delta X_r = \Delta Y_{r-1}$ and $p_r = \max_{\Delta Y_r}(\Delta X_r, \Delta Y_r)$.
if $p_1 \times p_2 \times \cdots \times p_r \geq B_r$ **then**
  $B_r = p_1 \times p_2 \times \cdots \times p_r$
end if
Return to the upper **procedure**.

2.4 Security Against Differential and Linear Cryptanalysis

Differential and linear cryptanalysis are the most powerful attacks against block ciphers. Since their invention and application on the DES cipher, two approaches have been followed to resist against these two attacks [79]. One approach concerns with the improvement of the overall structure of a block cipher in order to resist differential and linear attacks. The other approach improves the resistance provided by the cipher components. Here we discuss the component approach, for more details about the structure approach we refer the reader to [79].

As the round function of most modern block ciphers consist of a linear component (linear transformation) and a non linear component (Sboxes), designers tend to make the probabilities of differential and linear characteristics as small as possible by selecting an appropriate Sbox as well as an appropriate linear transformation. An appropriate Sbox is a one whose linear and differential approximations have the smallest probabilities among all the possible Sboxes of the same size. An appropriate linear transformation is a one that has a good diffusion in the sense that every output bit depends on all the input bits after a few number of rounds. Designers also give bound to the probability of a differential or linear characteristic by counting the minimum number of the so-called *active sboxes* which are the set of Sboxes that a non-zero input difference of the differential characteristic or a non-zero input mask of the linear characteristic passes through them. Clearly, the linear transformation together with the differential and linear properties of the chosen Sbox determines the number of active Sboxes.

The most notable design approach that bounds the probability of a differential and linear characteristic in a few number of rounds is the *wide trail strategy* [43] which was used in the design of the AES. The *wide trail strategy* simply consists of choosing the best possible Sbox together with designing a linear transformation layer that prevents the existence of differential and linear characteristics with low hamming weight. This results in increasing the number of active Sboxes in a few number of rounds and thus lowering the probabilities of differential and linear characteristics.

2.5 Other Attack Variants

Many other attacks exist on both block and stream ciphers. In the following we briefly describe the slide attacks and the algebraic attacks.

Slide Attacks: The slide attacks [16] were proposed to attack iterated block ciphers regardless of the number of rounds that they use. Slide attacks target ciphers that use identical round functions accompanied with a simple key schedule that uses identical round keys or alternates between two keys every round. Let F be the round function of the target block cipher. Then slide attacks look for two known plaintext/ciphertext pairs (P_1, C_1) and (P_2, C_2) such that $P_2 = F(P_1)$. If we find a P_2 such that $P_2 = F(P_1)$, then consequently, $C_2 = F(C_1)$. Such pairs are called slid pairs. Now if F is weak then the key could be derived from the two relations $P_2 = F(P_1)$ and $C_2 = F(C_1)$.

Algebraic Attacks: These kind of attacks exploit ciphers that can be expressed by a sparse multivariate non-linear equation system over the binary field $\mathbb{F}_2$ with the secret key bits as unknown variables and the plaintext/ciphertext pairs as the known variables. In general, solving random multivariate nonlinear system of equations is an NP-hard problem [51]. However, some techniques depending on the structure of the nonlinear system under consideration such as Gröbner basis and linearization are used to solve the nonlinear system in a reasonable time. While the Gröbner basis approach can sometimes take a reasonable amount of time, the Gröbner basis algorithms are hard to predict and in the worst case they have exponential complexity. The linearization approach replaces all the nonlinear terms in the equations with new independent variables in order to get a system of linear equations that have enough linearly independent equations which could be solved using Guassian elimination algorithm. If the new system is solvable, the attacker would then substitutes results back into the original system of equations to yield the secret key bits. Such simple linearization techniques were applied successfully against some stream ciphers [40].

Chapter 3

Lightweight Cryptography

Lightweight cryptography deals with the design of cryptographic primitives that are suitable to fit and run on small hardware devices such as RFID tags, sensor networks and contactless smart cards. Before the era of lightweight primitives, the landmark paper [50] proposes a low-cost implementation of the AES that costs around 3595 GE which is expensive for low-cost small devices [1]. This might indicate that implementing standard cryptographic algorithms on highly constrained devices is not an easy task.

The cryptographic community has pursued two directions to solve this problem: the first direction designs new solutions based on new cryptographic algorithms and protocols for lightweight devices. One solution in this direction belongs to the HB family which is a set of symmetric key challenge-response authentication protocols whose security is based on the hardness of the Learning Parity with Noise (LPN) problem [65, 71]. Another solution is the one-way function SQUASH which is a lightweight MAC proposed by Shamir [105]. The second direction designs lightweight cryptographic algorithms by reducing the state size and the hardware cost of the linear and nonlinear components of the cryptographic algorithm without making any compromise in security.

Thus, the challenge is in coping with the tradeoffs among security, hardware cost and performance. Saving area and power consumed by the cryptographic algorithm but at the same time avoiding any compromise concerning the security of the cryptographic algorithm. In this thesis, we are concerned with analyzing lightweight solutions that go inline with the second direction.

In this chapter, we give a short description about the lightweight ciphers that we studied in this thesis, namely, PRESENT, PRINTcipher and A2U2. We also give an overview about some of the attacks reported on them.

[1] A very compact implementation has been recently proposed in [89] requires 2400 GE

3.1 Lightweight Ciphers

Few years later after the publication of [50], many ciphers were developed for minimal hardware requirements. These lightweight ciphers have a small state or block size that are usually less than or equal 64. They also use a smaller key size compared to the standard encryption algorithms. Most of them use an 80 bit master key. In the following, we give a short description of the three ciphers that we investigated in this thesis. Namely, the block ciphers PRESENT, PRINTCIPHER and the stream cipher A2U2. We also investigated the core permutation of SPONGENT-88 which can be seen as a block cipher with a zero key xored with a round constant.

3.1.1 A Short Description of PRESENT and SPONGENT

One of the most prominent lightweight ciphers is the block cipher PRESENT proposed in 2007 [23]. It is an SPN consisting of 31 rounds. It has a 64-bit block size and uses an 80-bit master key (recommended for lightweight applications) or 128-bit master key. PRESENT-80 requires 1570 GE. The PRESENT round function consists of three layers (See Algorithm 4 and Figure 3.1), namely, the key addition layer, the Sbox layer (the nonlinear layer) and the permutation layer (the linear layer). The key addition layer simply xors the current state with the round key generated by the PRESENT key schedule (for more details about the key scheduling we refer to [23]).

The Sbox layer applies 16 times in parallel a single 4-bit Sbox. The PRESENT Sbox is described in the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Beside being chosen to be well-suited for efficient hardware implementation, the PRESENT Sbox has been chosen to fulfill several conditions to strengthen PRESENT against differential and linear cryptanalysis. More precisely, the PRESENT Sbox was chosen from a set of Sboxes, $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$, whose difference approximation table (as defined in Chapter 2) fulfills the following two conditions:

1. The probability that a nonzero input difference $\Delta X \in \mathbb{F}_2^4$ has a nonzero output difference $\Delta Y \in \mathbb{F}_2^4$, $\Pr(\Delta X, \Delta Y) \leq \frac{1}{4}$ for all ΔX and ΔY.
2. For any nonzero input difference $\Delta X \in \mathbb{F}_2^4$ and any nonzero output difference $\Delta Y \in \mathbb{F}_2^4$ with Hamming weight 1, $\Pr(\Delta X, \Delta Y) = 0$.

and whose linear approximation table (as defined in Chapter 2) fulfills the following two conditions:

1. The correlation of approximating $\langle b, S(x)\rangle$ by $\langle a, x\rangle$, $C(a,b) \leq \frac{1}{2}$.
2. For all the nonzero input masks $a \in \mathbb{F}_2^4$ and all the nonzero output masks b with Hamming weight 1, $C(a,b) = \pm\frac{1}{4}$.

The Permutation layer uses a bitwise permutation (See Figure 3.1) which costs only wires and 0 GE. The bitwise permutation is chosen so that full dependency (each output bit is dependent on all the input bits) is gained after a minimal number of rounds.

As noted in [77], for an SPN with block size b and s bit Sboxes, where s divides b, the following bit permutation

$$P(i) = \begin{cases} \frac{b}{s} \times i \bmod b-1 & \text{for } 0 \leq i \leq b-2, \\ b-1 & \text{for } i = b-1, \end{cases}$$

and also its inverse

$$P^{-1}(i) = \begin{cases} s \times i \bmod b-1 & \text{for } 0 \leq i \leq b-2, \\ b-1 & \text{for } i = b-1, \end{cases}$$

provide optimal diffusion as full dependency is gained after $\lceil \log_s b \rceil$ rounds.

In particular, PRESENT uses the above $P(i)$ with $b = 64$ and $s = 4$ and thus it gains full dependency after 3 rounds whereas PRINTCIPHER uses the inverse of $P(i)$ as its bitwise permutation as we will see later in its description. The significance of these optimal permutations is in thwarting statistical attacks with a smaller number of rounds compared to the case where non-optimal permutations are used. Constructing or finding these optimal permutations is not a trivial task as they are very rare. For more information about constructing optimal bitwise permutations we refer the reader to [82].

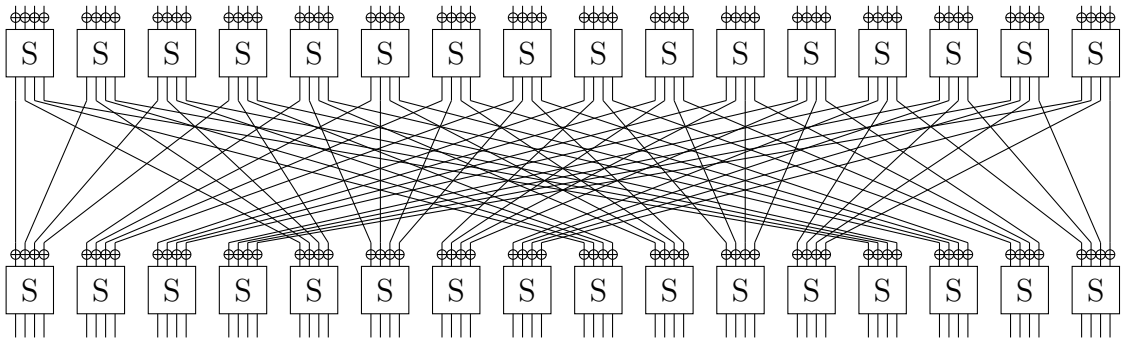
The simple and neat structure of PRESENT allowed its designers to give upper bounds on the probability of a differential characteristic and the bias of a linear characteristic. More specifically, the probability of a 4-round differential characteristic is upper bounded by 2^{-20} and thus the probability of a 25-round is upper bounded by $2^{-100}(\leq 2^{-64})$ and the absolute correlation value of a 4-round linear characteristic is upper bounded by 2^{-6} and thus the absolute value for a 28-round linear characteristic is upper bounded by $2^{-42}(\leq 2^{-32})$.

SPONGENT is a new lightweight hash function [22], its core permutation is inspired by PRESENT as it inherits its three layers. There are many variants of SPONGENT depending on its hash size and the block size of its permutation.

Algorithm 4 Pseudo-code of the PRESENT cipher

Require: Plaintext $= P$, $K = (K_0, K_1, \cdots, K_r)$
Ensure: $C = E_K(P)$
1: STATE $\leftarrow P$
2: **for** i=0:r-1 **do**
3: STATE $\leftarrow$ STATE $\oplus K_i$
4: STATE $\leftarrow$ SboxLayer(STATE)
5: STATE $\leftarrow$ PLayer(STATE)
6: **end for**
7: STATE $\leftarrow$ STATE $\oplus K_r$

Fig. 3.1: One round of the block cipher PRESENT.



The permutation layer of SPONGENT is the same as PRESENT except that it is wider. The SPONGENT Sbox has been chosen according to the same criteria of the PRESENT Sbox. It has been chosen carefully to avoid the existence of many linear trails with only one active Sbox per round found on PRESENT. The SPONGENT Sbox is described in the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

The core permutation of SPONGENT can be seen as a cipher using identical round keys (the key is almost the zero key with few bits at the leftmost and the rightmost generated from a counter that is meant to prevent sliding properties and invariant subspaces).

In other words, the core permutation of SPONGENT can be seen as a PRESENT-like cipher which is a definition we borrowed from [27]. Thus the core permutation of SPONGENT can then be represented by the above pseudo-code of PRESENT using zero round keys xored with the corresponding round constant. For more details about the description of PRESENT and SPONGENT, we refer to [22, 23].

3.1.2 A Short Description of PRINTcipher

PRINTcipher is a PRESENT-like block cipher proposed in 2010 [77]. It has two variants PRINTcipher-48 and PRINTcipher-96 mainly designed to exploit the properties of IC-printing technology which targets the fabrication of cheap RFID tags. PRINTcipher-48 and PRINTcipher-96 require only 402 GE and 726 GE respectively.

To reduce the size of the implementation, PRINTcipher uses a 3-bit Sbox that costs less than 12 GE but at the cost of having more rounds to thwart differential and linear cryptanalysis. It also uses a small block size, only 48 bits, in its smallest variant, Moreover, it uses a fixed key since it is very unlikely that the secret key of an RFID tag will be changed.

Regular IC technology allows all versions of the cipher to be identical and as a post-fabrication step it personalizes each RFID tag with a unique key. However, with IC-printing technology there is a little cost in changing the circuit that is printed at each run. Thus, part or all of the secret key can be embedded into the algorithm description.

PRINTcipher-48 (resp -96) is a PRESENT-like cipher with a block size of $b = 48$ (resp $b = 96$) bits and 48 (resp 96) rounds. The key size is 80 bits for PRINTcipher-48 and 160 bits for PRINTcipher-96. PRINTcipher uses a single 3 bit Sbox shown in the following table.

x	0	1	2	3	4	5	6	7
$S[x]$	0	1	3	6	7	4	5	2

In the non-linear layer the current state is split into 16 words of 3 bits for PRINTcipher-48 and into 32 words of 3 bits for PRINTcipher-96 and each word is processed by the Sbox in parallel. The linear layer consists of the inverse of the PRESENT permutation with $b = 48$ or $b = 96$ and $s = 3$.

The peculiar part of PRINTcipher is to have all rounds identical up to adding a round constant on a small number of bits. Here identical has to be understood as including the round key, in other words, all round keys are identical. As a simple round key xored to the state in each round limits the key size to 48 resp 96 bits, an additional key-dependent permutation layer was introduced. This permutation layer permutes the input bits of each Sbox individually. Out of 6 possible permutations on 3 bits, only four are valid permutations for PRINTcipher.

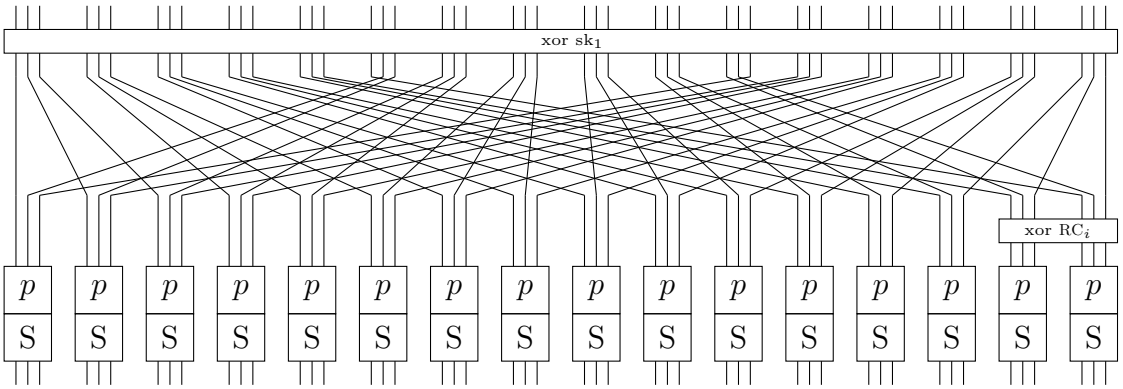
For PRINTcipher-48 the 80-bit user-supplied key k is split into two subkeys $k = \mathrm{sk}_1||\mathrm{sk}_2$ where sk_1 is 48 bits long and sk_2 is 32 bits long. The first subkey sk_1 is xored to the state at the beginning of each round. The second subkey sk_2 is used to generate the key-dependent

permutations in the following way. The 32-bits are divided into 16 sets of two bits and each two-bit quantity $a_1||a_0$ is used to pick one of four of the six available permutations of the three input bits. Specifically, the three input bits $c_2||c_1||c_0$ are permuted to give the following output bits according to two key bits $a_0||a_1$.

$a_1\|\|a_0$	
00	$c_2\|\|c_1\|\|c_0$
01	$c_1\|\|c_2\|\|c_0$
10	$c_2\|\|c_0\|\|c_1$
11	$c_0\|\|c_1\|\|c_2$

One round of PRINTCIPHER-48 is shown in Figure 3.2.

Fig. 3.2: One round of PRINTCIPHER-48 illustrating the bit-mapping between the 16 3-bit Sboxes from one round to the next. sk_1 denotes the xor key, p the permutation key, and RC_i the round counter.



3.1.3 A Short Description of A2U2

The stream cipher A2U2 was presented at IEEE RFID 2011 [46]. It has a key length of 56 bit. The cipher's inner state consists of a counter LFSR C (7 bits), two non-linear feedback shift registers A and B (NFSRs, 17 and 9 bits), and a key register K (56 bits). Note that we use a different notation than in the original paper, ours being more suitable for cryptanalysis.

Even though this is not apparent from the original paper [46], the authors confirmed upon request that the cipher first outputs and then updates its inner state. In the following, the opposite order is used, though purely for didactical reasons.

Updating the Counter: In the following, we denote the state of the counter LFSR at time t by $C^t = (C_t, C_{t-1}, \ldots, C_{t-6})$, for $t = 0, 1, \ldots$. The starting state after initialization is $C^0 = (1, 1, \ldots, 1)$, see below under "Initialization". The LFSR then uses the feedback recurrence

$$C_t = C_{t-7} + C_{t-4}$$

for updating the state for $t \geq 1$. It is an m-LFSR, i.e. it has maximal period (i.e., $2^7 - 1$).

Updating the NFSRs: We denote the state of the NFSRs by $A^t = (A_t, \ldots, A_{t-16})$ and by $B^t = (B_t, \ldots, B_{t-8})$. The update uses an auxiliary variable h_t (defined below under "Updating the key register") and the following non-linear feedback recurrences:

$$\begin{aligned} B_t &= A_{t-17} + A_{t-15}A_{t-14} + A_{t-12} + A_{t-10}C_{t-7} + A_{t-7}A_{t-6}A_{t-5} + A_{t-4}A_{t-2} \\ A_t &= B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + h_t + 1, \end{aligned}$$

again for $t \geq 1$.

Updating the Key Register The key register is a rotation register, i.e. the state in time t is a rotated version of the initial state. If we denote the key bits by $(k_0, \ldots, k_{55})$, then each state of the key register is defined as

$$K^t = (k_{5t}, k_{5t+1}, \ldots, k_{5t+55}),$$

where all indices are computed modulo 56.

For each round, the first five bits of the register are stored in a buffer $S^t = (S_0^t, \ldots, S_4^t) = (k_{5t}, \ldots, k_{5t+4})$, and they are used to compute the auxiliary variable h_t as follows:

$$h_t = \mathrm{MUX}_{C_{t-5}}(S_0^t, S_1^t) \cdot \mathrm{MUX}_{C_{t-1}}(S_4^t, A_{t-2}) + \mathrm{MUX}_{C_{t-3}}(S_2^t, S_3^t) + 1,$$

where $\mathrm{MUX}_z(x, y)$ is the multiplexer function that selects x if $z = 0$ and y otherwise.

Initialization: The A2U2 cipher has a 61-bit key split into two parts: A 5-bit "counter key" and a 56-bit "register key". In addition, the cipher receives a 32-bit random number (which are only 31 bits used of). Key and random number are written into the registers as follows:

- **Counter Register:** The 5 least significant key of the 61-bit key are used as the "counter key" and are bitwise xored with the 5 least significant random bits. The resulting 5-bit

vector is written into the 5 least significant bits of the counter LFSR (using the above notation). The second MSB is set to 1, the MSB is set to 0.

- **NFSRs:** The next 26 key and random bits are bitwise xored and written into the NFSR cells.
- **Key Register:** The 56 register key bits (to some extent the same that were used for NFSR initialization) are stored in the key register.

Now the cipher is clocked until the counter register reaches the all-one state. This happens after 9-126 clockings. The resulting state is called the initial state.

Output Generation: The cipher deploys a form for irregular output mechanism; it outputs either encrypted plaintext bits or pseudo-random bits depending on the content of NFSR cell A_t. Plaintext bits have to "wait" until $A_t = 1$ before being encrypted. If we denote the plaintext string by $P = (P_0, P_1, \ldots)$ and if we define $\sigma(t) = \sum_{i=0}^{t-1} A_i$ with $\sigma(0) = 0$, then the output of the cipher in round t is:

$$Y_t = \mathrm{MUX}_{A_t}(B_t + C_t, B_t + P_{\sigma(t)}).$$

3.2 Other Lightweight Primitives

There are many lightweight primitives proposed in the literature. In the following, we give a brief description about some of them.

There are two stream ciphers suitable for lightweight applications emerged from the eStream project. Namely, Grain [60] and Trivium [32]. Grain has a compact hardware implementation. It consists of two variants. One uses an 80-bit key with a 80-bit initialization vector (where 16 bits are fixed to 1) which is the recommended variant. The other uses 128-bit key with an 128-bit initialization vector (where 16 bits are fixed to 1). Both of its variants consists of a LFSR initialized by the IV, a NFSR initialized by the secret key bits and a nonlinear output Boolean function that takes some inputs from the LFSR and the NFSR. Grain-80 has 160 initialization rounds and Grain-128 has 256 initialization rounds. During the initialization rounds no output is produced but is fed back into the LFSR and the NFSR. The structure of the two variants of Grain looks very similar but each of their components use a different Boolean function. For instance, the NFSR of Grain-80 uses a nonlinear update function with algebraic degree 6 whereas the NFSR Grain-128 uses a nonlinear update function with algebraic degree 2 which makes its full version suspectable to the recent dynamic cube attacks for some weak keys and

this has led to modifying Grain-128 to Grain-128a. The hardware implementation of Grain-80 suggests a cost of 1294 GE [54] which makes it well-suited for lightweight applications.

Trivium has a very simple design. It uses an 80-bit secret key together with an 80-bit initialization vector. Its internal state uses 288-bit and it consists of three interconnected NFSRs of different lengths where each use a quadratic Boolean function. It uses 1152 initialization rounds before producing any keystream bits. The hardware implementation of Trivium suggests a cost of 2580 GE [54].

In 2009, a block cipher family called KATAN/KTANTAN was proposed [31]. It is a family of block ciphers inspired by the stream cipher Trivium. It uses two NFSRs that update each other. It has three variants with small block sizes, namely 32-bit, 48-bit and 64-bit. All these variants use an 80-bit key and 255 rounds. The only difference between KATAN and KTANTAN is in the key scheduling as in KTANTAN the key is fixed and cannot be changed.

In 2010, QUARK the first lightweight sponge-based hash function was proposed [9]. Its design is similar to that of KATAN and Grain and its number of rounds exceed 255. It has three variants depending on its output size together with the input size of its core permutation. Its smallest variant fits in 1379 GE.

In 2011, lightweight AES-like primitives has been proposed. Namely, the PHOTON hash function [56] and the LED block cipher [57] were proposed. Their diffusion layers use an MDS matrix which can be efficiently computed in a serial way. The smallest variants of PHOTON and LED require 865 and 966 GE respectively. Also in 2011, the block cipher Piccolo, a generalized Feistel block cipher, was proposed [107]. Similar to PHOTON and LED, Piccolo diffusion layer also uses an MDS matrix. Its smallest version requires less than 700 GE.

3.3 Overview of Some Attacks in PRESENT, PRINTcipher and A2U2

In this section, we briefly describe some of the attacks on the ciphers we analyzed in this thesis, namely, PRESENT, PRINTcipher and A2U2.

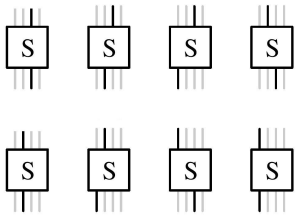
3.3.1 Attacks on PRESENT

The PRESENT block cipher has been analyzed in several publications. In [109], a 16-round differential attack was mounted. Later a statistical saturation attack was mounted and claimed to break 24 rounds [39]. In [93], the author showed the existence of 32% weak keys which have

a higher bias that makes the cipher using those weak keys distinguishable for up to 24 rounds. In [38], a multidimensional attack was used to break 25 rounds and 26 rounds where the latter attack used the whole code book. Recently, a multiple differential attack was mounted on 18-round of PRESENT [18]. Linear attacks were more successful in breaking more rounds in PRESENT than differential attacks. This is due to bitwise permutation of PRESENT together[2] with the following two facts about the PRESENT Sbox:

- The fact that the design criteria of the PRESENT Sbox allows the existence of linear characteristics with one active Sbox per round and prevents the existence of differential characteristics with one active Sbox per round.
- The existence of eight linear approximations in the PRESENT Sbox with input and output masks of Hamming weight one (See the linear approximations of the PRESENT Sbox in the Figure below and Table 3.1).

Fig. 3.3: Biased 1-1 bit approximations in the PRESENT Sbox



Ohkuma exploited the second fact to find linear approximations with input and output masks of Hamming weight one that consists of many linear characteristics activating only one Sbox with the same absolute bias. Considering these many linear characteristics and assuming independent round keys, Ohkuma showed that the bias distribution over the entire key space of some r-round linear approximations of input and output masks of Hamming weight one follows a normal distribution with average 0 and variance $2^{(-2r)^2}N$, where N is the number of linear trails activating one Sbox per round. Therefore, 32% of the keys in the key space have an absolute correlation $\geq 2^{-2r}\sqrt{N}$. Such keys were defined as weak keys. Ohkuma showed that for 28-round PRESENT there are linear approximations with absolute correlation $2^{-38.3} < 2^{-42}$ for some weak keys. This is higher than the 2^{-42} upper bound for the absolute correlation value of a linear characteristic set by the designers for 28-round PRESENT. However, as the

[2]It was observed in [82] that given the PRESENT bit permutation, the PRESENT Sbox is among the 8% worst of all optimal Sboxes. Also given the PRESENT Sbox, the PRESENT bit permutation was the worst among 2^{21} chosen optimal permutations.

Table 3.1: Linear Approximation Table of the PRESENT Sbox. The first column holds the integer represenation of the binary input masks while the first row holds the integer representation of the binary output masks. The entry (i, j) divided by 16 represents the correlation of a linear approximation with input mask equal to the binary representation of i and output mask equal to the binary representation of j. 1-bit to 1-bit masks are marked with boxes.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	-8	0	-8	0	0	0	0	0	-8	0	8
2	0	0	4	4	-4	-4	0	0	4	-4	0	8	0	8	-4	4
3	0	0	4	4	4	-4	-8	0	-4	4	-8	0	0	0	-4	-4
4	0	0	-4	4	-4	-4	0	8	-4	-4	0	-8	0	0	-4	4
5	0	0	-4	4	-4	4	0	0	4	4	-8	0	8	0	4	4
6	0	0	0	-8	0	0	-8	0	0	-8	0	0	8	0	0	0
7	0	0	0	8	8	0	0	0	0	-8	0	0	0	0	8	0
8	0	0	4	-4	0	0	-4	4	-4	4	0	0	-4	4	8	8
9	0	8	-4	-4	0	0	4	-4	-4	-4	-8	0	-4	4	0	0
10	0	0	8	0	4	4	4	-4	0	0	0	-8	4	4	-4	4
11	0	-8	0	0	-4	-4	4	-4	-8	0	0	0	4	4	4	-4
12	0	0	0	0	-4	-4	-4	-4	8	0	0	-8	-4	4	4	-4
13	0	8	8	0	-4	-4	4	4	0	0	0	0	4	-4	4	-4
14	0	0	4	4	-8	8	-4	-4	-4	-4	0	0	-4	-4	0	0
15	0	8	-4	4	0	0	-4	-4	-4	4	8	0	4	4	0	0

designers' upper bound is for a linear characteristic and not for the linear approximation or linear hull, Ohkuma's result does not contradict the designers' 2^{-42} upper bound. Ohkuma used a 20-round linear approximation to break 24-round of PRESENT.

Table 3.2 shows the number of linear characteristics activating one Sbox that exist within a linear approximation with input and output masks of Hamming weight one. The table clearly shows that the correlations of the linear approximations exceed the designers' characteristic bound for large number of rounds which might be the reason why the experiments performed by the designers which were probably for small number of rounds have not shown the effect of the linear hull which seems to appear for large number of rounds. As shown in Table 3.2, when the number of rounds exceeds 11, the maximum squared average correlation of a linear approximation with input and output masks of Hamming weight one in PRESENT is larger than the maximum squared correlation of a linear characteristic.

Table 3.2: $r \equiv$ Number of rounds, $N \equiv$ number of linear trails, $C_1^2 \equiv$ maximum average squared correlation of the linear approximation with input and output masks of Hamming weight one in PRESENT and $C_2^2 \equiv$ maximum squared correlation possible for a linear characteristic in PRESENT.

r	$\log_2(N)$	$\log_2(C_1^2)$	$\log_2(C_2^2)$
4	1.58	-14.41	-12.00
...	...	...	...
8	7.58	-24.42	-24.00
...	...	...	...
11	7.58	-24.42	-24.00
12	18.73	-34.82	-36.00
...	...	...	...
20	24.29	-55.71	-60.00
...	...	...	...
23	28.45	-63.55	-68.00
24	29.84	-66.16	-72.00
...	...	...	...
28	35.39	-76.61	-84.00

The multidimensional linear attack on PRESENT mounts a 25-round attack using a 23-round distinguisher with multiple linear approximations activating one Sbox per round but having input and output masks with Hamming weight more than or equal to one. A 26-round attack was mounted using the whole code book using a 24-round distinguisher with multiple linear approximations activating one Sbox per round but having input and output masks with Hamming weight more than or equal to one.

These linear attacks have influenced the design of SPONGENT since it uses the same but extended bitwise permutation of PRESENT as its linear layer and the same criteria of the PRESENT Sbox. Thus to avoid the threats of such attacks, the SPONGENT Sbox was carefully chosen to avoid the existence of many linear characteristics with one active Sbox per round as the case in PRESENT. Table 3.3 shows the existence of only 4 linear approximations with input and output masks with nonzero correlation in SPONGENT compared to 8 in PRESENT. This makes SPONGENT secure against linear attacks in fewer rounds compared to PRESENT. For instance in SPONGENT-88, there is always only one trail that have one active Sbox at each round when the number of rounds is larger than 5.

Table 3.3: Linear Approximation Table of the SPONGENT Sbox. The first column holds the integer representation of the binary input masks while the first row holds the integer representation of the binary output masks. The entry (i, j) divided by 16 represents the correlation of a linear approximation with input mask equal to the binary representation of i and output mask equal to the binary representation of j. 1-bit to 1-bit masks are marked with boxes.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	-4	4	4	-4	0	0	0	0	-4	4	-4	4	-8	-8
2	0	0	0	-8	0	0	0	-8	-4	-4	-4	4	4	4	4	-4
3	0	0	-4	-4	-4	4	8	0	4	4	0	0	0	8	-4	4
4	0	0	0	0	0	8	0	8	-4	-4	4	4	-4	4	4	-4
5	0	0	-4	4	-4	-4	8	0	-4	-4	0	-8	0	0	4	-4
6	0	0	-8	0	-8	0	0	0	0	0	0	8	0	-8	0	0
7	0	0	4	4	-4	-4	0	0	8	-8	4	4	4	4	0	0
8	0	0	0	0	0	0	0	0	0	8	8	0	8	0	0	-8
9	0	8	-4	-4	-4	-4	-8	0	0	0	4	-4	-4	4	0	0
10	0	0	0	-8	0	0	0	8	4	-4	-4	-4	4	-4	-4	-4
11	0	8	-4	4	4	4	0	0	-4	-4	0	0	8	0	-4	4
12	0	0	0	0	0	-8	0	8	-4	4	-4	4	4	4	4	4
13	0	-8	-4	-4	4	-4	0	0	-4	-4	8	0	0	0	-4	4
14	0	0	8	0	-8	0	0	0	-8	0	0	0	0	0	-8	0
15	0	8	4	-4	4	-4	8	0	0	0	4	4	-4	-4	0	0

In Chapter 4, we used the submatrix method to find better linear approximations in PRESENT and SPONGENT than the previously found linear approximations.

3.3.2 Attacks on PRINTcipher

The block cipher PRINTcipher has been analyzed in several publications. Our work in Chapter 5 presents the first analysis of PRINTcipher which basically showed how to avoid

the key dependent layer of PRINTcipher to mount a differential attack on PRINTcipher. All the subsequent attacks on PRINTcipher point to the existence of weak keys. Our work in Chapter 6 introduces the Invariant Subspace Attack which can be seen as a partial fixed point attack and applies it to PRINTcipher to distinguish the full round PRINTcipher for two classes of keys, each class contains 2^{-29} fraction of all the keys, we call these keys "weak keys". Both of these attacks exploits the difference properties of the 3-bit Sbox used in PRINTcipher. In the following, we show the difference distribution table of the PRINTcipher Sbox.

Table 3.4: Difference distribution table for PRINTcipher Sbox. Note that the difference table is symmetric. 1-bit to 1-bit differences are marked with boxes. Each entry divided by 8 gives the corresponding difference probability.

		Δy							
		000	001	010	011	100	101	110	111
Δx	000	8	0	0	0	0	0	0	0
	001	0	$\boxed{2}$	0	2	0	2	0	2
	010	0	0	$\boxed{2}$	2	0	0	2	2
	011	0	2	2	0	0	2	2	0
	100	0	0	0	0	$\boxed{2}$	2	2	2
	101	0	2	0	2	2	0	2	0
	110	0	0	2	2	2	2	0	0
	111	0	2	2	0	2	0	0	2

As mentioned in Chapter 6, a one bit difference in the input of the Sbox causes a one bit difference in the output difference with probability $\frac{2}{8}$. This allows the existence of some specified 1-bit input difference and 1-bit output difference with probability 1 such as those defined in Table 6.1. Since 1-bit input difference and 1-bit output difference with nonzero probability cannot be avoided in an optimal 3-bit Sbox [77], similar properties as the one showed in Table 6.1 would exist for any optimal 3-bit Sbox.

More recently, the work in [30] found all the classes of weak keys for both PRINTcipher-48 and PRINTcipher-96. More specifically, 64 classes of weak keys have been found for PRINTcipher-48 including the two classes mentioned in Chapter 6 and 115,669 classes of weak keys have been found for PRINTcipher-96 including the two classes mentioned in Chapter 6. However, for PRINTcipher-48 the two classes mentioned in Chapter 6 contain the largest number of weak keys which is 2^{51}. It is also noted in [30], that these two specific classes are not disjoint and contain 2^{45} joint elements. In general, the classes of weak keys found might have a non-trivial intersection.

Two methods were proposed in [30] to find all the classes of weak keys in PRINTcipher. The first method is based on a one-to-one correspondence between invariant subspaces in PRINTcipher-n and subsets in $\mathbb{Z}_n$ satisfying certain properties (refer to [30] for more details). It finds invariant subspaces in PRINTcipher-48 by finding subsets in $\mathbb{Z}_{48}$ that satisfies certain properties. The method goes through all the possible permutation keys, that allow the existence of invariant subspaces, which are 2^{28} (out of 2^{32} but 4 bits get affected by the round constants (See Chapter 6)). After finding the permutation keys that have invariant subspaces, the xor key is found. This method was able to find all the 64 classes of weak keys existing in PRINTcipher-48. However, for PRINTcipher-96 this method is infeasible since it exhaustively tries all the possible permutation keys, that allows the existence of invariant subspaces, which are 2^{60} (out of 2^{64}). Therefore, the second method was proposed. It is based on a one-to-one correspondence between PRINTcipher-n and points of a certain polytope in $\mathbb{Z}^n$. These points and thus the invariant subspaces in PRINTcipher are then found efficiently using Mixed Integer Linear Programming. For more details, we refer the reader to [30].

The work in [72] combines differential and linear attacks. More specifically, linear approximations were used to increase the probability of differential characteristics in order to break 29 rounds and 31 rounds of PRINTcipher-48 for 4.54% and 0.036% of the keys, respectively. In [6], linear attacks that break 28 rounds and 29 rounds of PRINTcipher-48 for 50% and 3.125% of the key space respectively were proposed.

3.3.3 Attacks on A2U2

In [36], a very efficient chosen plaintext attack against A2U2 is proposed. However, as we are going to show here, the attack contains a flaw that makes it unapplicable against the real A2U2 cipher.

Attack Idea: If the attacker could freely choose one plaintext bit for each clock, then he can write the output equation as follows:

$$\begin{aligned} Y_t &= \mathrm{MUX}_{A_t}(B_t + C_t, B_t + p_t) \\ &= \mathrm{MUX}_{A_t}(C_t, p_t) + B_t. \end{aligned}$$

Depending on the amount of knowledge the attacker has about the plaintext, he can now learn more about the inner state.

If the attacker can choose the plaintext, he can start by encrypting a plaintext that is identical

to the counter sequence. In this case, the above equation simplifies to

$$\begin{aligned} Y_t &= \mathrm{MUX}_{A_t}(C_t, p_t) + B_t \\ &= p_t + B_t, \end{aligned}$$

meaning that the attacker can learn the whole sequence $(B_t)_{t\geq 0}$.

Next, he encrypts a plaintext that is the bitwise inverse of the counter sequence. This allows him to distinguish for every ciphertext bit whether C_t or p_t was encrypted, providing the attacker with the full sequence $(A_t)_{t\geq 0}$.

Now he has the sequences produced by the LFSR and by both NFSRs. All that remains is to test for each round which key bit gives the correct NFSR update. This can be done in unit time, yielding an extremely efficient attack.

The Catch: However, the initial assumption of the above attack is wrong, invalidating the whole cryptanalysis. The problem is that plaintext is not used at a rate of 1 bit per round. It is not possible to choose a plaintext bit for each round, because (1) some plaintext bits are used in several rounds and (2) without knowledge of the sequence $(A_t)_{t\geq 0}$, it is impossible to say in which rounds a plaintext bit will be used. However, as it turns out, this attack can be repaired. In Chapter 7, we show how to repair this attack besides providing other attacks.

Since the first version of this paper, this problem was acknowledged by Chai et al., and their paper was updated accordingly[3]. Their new attack recovers first the sequence $(A_t)_{t\geq 0}$ by choosing two complementary plaintexts (p_t and its complement $\bar{p_t}$) and xoring their corresponding ciphertexts ($c \oplus \bar{c} = \Delta c$). Now, if $\Delta c_t = 0$ then $A_t = 0$, otherwise $A_t = 1$. This recovers the sequence $(A_t)_{t\geq 0}$ and consequently the sequence $(B_t)_{t\geq 0}$ can then be simply recovered.

More recently, the work in [111] improved the guess-and-determine known plaintext attack in Chapter 7. The main idea of [111] is in forming the recurrence relation

$$\sigma(i) = \sigma(i+1) - A_i \qquad \text{and} \qquad 0 \leq \sigma(i) \leq i$$

where $0 \leq i \leq n$ and n is the number of the ciphertext bits. The authors of [111] reduced the computational time for determining registers A and B by guessing $\sigma(n)$ (which is $\leq n$

[3]A word on the chronological ordering of things seems to be in order here. The original Chai attack was published on IACR Eprint May 17, 2011. Our attack proposed in Chapter 7 was submitted to Eprint May 25, 2011. On the same day, a copy was sent to Chai et al., informing them about a possible flaw in their attack and proposing the attack described below. On May 26, 2011, Chai et al. updated their Eprint paper by removing the flaw and proposing a new attack similar (but not identical) to the one the we submitted the day before.

and thus a small number) in addition to guessing register A and then determining previous states of registers A and B. The paper also shows by experiments that the quadratic equation system obtained after determining the subkeys can easily be solved and most of the time there is always a one key candidate. All these takes time complexity in the order of $2^{24.7}$.

Chapter 4

Differential and Linear Approximations on PRESENT-Like Ciphers

The lightweight block cipher PRESENT described in Chapter 3 (See Section 3.1.1) has received a lot of attention from the cryptographic community and it has been recently adopted by ISO as one of the international standards in lightweight cryptography [67]. PRESENT has inspired some lightweight designs, namely, the block cipher Maya [52], the involution block cipher PUFFIN [37][1], the block cipher PRINTcipher (See Chapter 3) and the core permutation of the hash function SPONGENT (See Chapter 3). All these designs can be seen as PRESENT-like ciphers (which is a definition we borrowed from [27]) as their algorithms are very similar to the Algorithm of PRESENT (See Algorithm 4).

In this chapter we show how to find good differential and linear approximations on PRESENT-like ciphers. We also study the effect of key scheduling on the distribution of differential and linear approximations over all the key space.

In the first section, we present an approach to find low-weight differential and linear approximations on PRESENT-like ciphers and then apply it on PRESENT and SPONGENT. In the second section, we study the effect of key scheduling on the distribution of differential and linear approximations on PRESENT-like ciphers. In the final section, we conclude and give some suggestions for future work.

[1]Note that both Maya and PUFFIN were completely broken using both differential and linear cryptanalysis in [27, 28] and [20, 82] respectively. Therefore we do not consider them here.

4.1 Estimating the Probabilities of Low-Weight Differential and Linear Approximations

We estimate the probability of low-weight linear and differential approximations in PRESENT-like ciphers. By using large but sparse correlation and difference submatrices, we overcome the memory and time problems that appears in the branch and bound method [87] described in Algorithm 2.3 when the number of good linear and differential trails grows exponentially. For instance, there would be memory and time problems in the branch and bound method when the number of differential and linear trails grows exponentially but using sparse correlation and difference submatrices we can handle any number of trails and investigate many differential and linear approximations with a negligible cost. For instance, all the linear approximations of PRESENT at Table 4.2 come from a number of good and bad trails exceeding 2^{89} which clearly shows a convincing advantage of using sparse submatrices over the branch and bound algorithm when the number of trails grows exponentially.

Using sparse correlation and difference submatrices of PRESENT and SPONGENT:

1. We report the first improved analysis on SPONGENT [22], specifically we improve the linear cryptanalysis on the SPONGENT permutation presented by the designers by one more round.

2. We present better linear approximations on PRESENT and present a 24-round statistical saturation distinguisher that uses better input and output subspaces compared to the original attack paper [39]. These approximations also show that the assumption, made by all the previous analyses on PRESENT [38, 82, 93], that the linear approximations consisting of trails with one active Sbox at each round, yield the highest bias is not valid. We also found many 16-round differential approximations activating at most 4 Sboxes per round with probability larger than 2^{-64}, which could be used to mount a differential attack on 18-round PRESENT similar to the ones in [18, 109].

4.1.1 Description of Our Estimation Approach

Assuming that PRESENT-like ciphers are Markov ciphers [81,91], we make use of submatrices of the correlation and the transition probability matrices of the target ciphers to find the best linear and differential approximations. We focus only on describing how to find better linear approximations. The case for finding better differentials is similar (use transition probability

submatrix instead of correlation submatrix and input-output difference instead of input-output masks).

Large Sparse Correlation and Difference Matrices

In [21, 22], a submatrix of the correlation matrix of size $4n_s \times 4n_s$ was used to estimate the correlation of a linear approximation, where n_s is the number of the 4-bit Sboxes used in the permutation where only input and output masks of Hamming weight one are considered. We extend this approach by adding input and output masks with Hamming weight ≤ 4. This results in having a large correlation submatrix whose entries activate at most 4 active Sboxes. Suppose that we use a correlation submatrix with input and output masks with Hamming weight less than or equal to m. Then the size of the submatrix will be $\sum_{i=1}^{m} \binom{n}{i} \times \sum_{i=1}^{m} \binom{n}{i}$, where n is the block size of the cipher in bits. The submatrix size is large but most of its entries are zeros. For instance, we see that for any input element activating 's' Sboxes ($1 \leq s \leq m$), all the other output elements corresponding to the other Sboxes yield a zero correlation. Thus, there are more than

$$\sum_{i=1}^{m} \binom{n}{i} - 15^s$$

zero output elements for any input activating 's' Sboxes. Thus, this submatrix has few non zero elements and therefore it can easily fit in memory using a sparse matrix storage format (See Section 4.2).

The construction of the correlation submatrix is straightforward. For instance to fill the submatrix entries from an input with Hamming weight 5, we proceed as follows: for each possible input, we determine the number of activated Sboxes which is in this case at least 2. Suppose we have i active Sboxes, then all the possible ordered solutions of the inequality $x_1 + x_2 + \cdots + x_i \leq m$ determine the Hamming weight of the output bits of each of the i active Sboxes. Then we fill the submatrix entries corresponding to the specified input by considering all the possible output bits of the specified Hamming weight. To estimate the cost of filling these entries, we consider the simple case where we have two active Sboxes. The time cost for filling the corresponding submatrix entries is

$$\sum_{2 \leq i+j \leq 5} \binom{4}{i} \binom{4}{j}$$

and the number of all the possible inputs with Hamming weight 5 activating 2 Sboxes is

$$N_2 = \sum_{w_1+w_2=5} \binom{\frac{n}{4}}{2} \binom{4}{w_1} \binom{4}{w_2}.$$

Now by symmetry, the cost of filling the corresponding entries that have outputs with Hamming weight 5 activating 2 Sboxes is similar but we only exclude the duplicated cases where $i+j=5$, so the cost is $\sum_{2\leq i+j<5}\binom{4}{i}\binom{4}{j}$. Therefore, the total construction time of input and output with Hamming weight 5 activating 2 Sboxes is

$$2N_2\sum_{2\leq i+j\leq 5}\binom{4}{i}\binom{4}{j}-N_2\sum_{i+j=5}\binom{4}{i}\binom{4}{j}.$$

One can see that the construction time can be generalized as follows.

Proposition 4.1.1. *The time cost for computing the correlations corresponding to inputs and outputs of Hamming weight 'w', $1\leq w\leq m$ is in the order of:*
$2(N_1\sum_{1\leq i\leq w}\binom{4}{i}+N_2\sum_{2\leq i+j\leq w}\binom{4}{i}\binom{4}{j}+\cdots+N_{w-1}\sum_{w-1\leq i+\cdots+z\leq w}\binom{4}{i}\cdots\binom{4}{z})$
$+N_w4^w-N_2\sum_{i+j=w}\binom{4}{i}\binom{4}{j}-\cdots-N_{w-1}\sum_{i+\cdots+z=w}\binom{4}{i}\cdots\binom{4}{z}$, *where* $N_1+\cdots+N_w=\binom{n}{w}$ *and* $N_i=\sum_{w_1+\cdots+w_i=w}\binom{n}{i}\binom{4}{w_1}\cdots\binom{4}{w_i}$ *is the number of elements with Hamming weight 'w' activating 'i' number of Sboxes.*

Note that $N_1=0$ when $w\geq 5$ as in this case we have at least two active Sboxes. The total construction time is in the order of the sum of construction times of all the possible input and output weights (w), i.e. $1\leq w\leq m$, where the dominant term is when $w=m$.

After constructing the correlation submatrix, C. The correlation approximations after r rounds is computed by $C^r=\prod_{i=1}^{r}M_i$, where M_i is the correlation submatrix at round i formed by changing the signs of C according to the round key and the round constant used in the cipher. The maximum correlation after r rounds is thus given by $c^r_{max}:=\max|C^r_{ij}|$. This works in the case of SPONGENT since we know that it uses an almost zero key at each round and thus we can compute the actual correlation of each approximation but in the case of PRESENT, we need to compute the average squared correlation of a linear approximation (See Proposition 4.2.2) in order to compute the capacity of the statistical saturation attack.

As the size of the correlation submatrix gets bigger when considering masks with Hamming weight equal to 4, the matrix-matrix multiplications might not be always possible for high number of rounds especially when there are many trails for most of the approximations as these make the resulted submatrix C^r very dense and consequently we might run out of memory. Thus, instead we use successive matrix-vector multiplications as described by Algorithm 5.

Note that before Step 3 in Algorithm 5 when we are computing the maximum absolute correlation (for example in SPONGENT), we have to change the signs at some entries of the correlation submatrix M at each round according to the corresponding round constant. The time complexity of Algorithm 5 is in the order of $l\times(r-1)$ matrix-vector multiplications where l is the number of rows or columns of the submatrix. If l is a large number, then the

Algorithm 5 Finding the best the average squared correlation (absolute correlation)

Require: Submatrix M of size $l \times l$, M is a submatrix of the average squared correlation matrix (or the correlation matrix).

Require: Two temporary vectors of length "l", tempCorr and tempIndex.

Ensure: Finds the best r-round average squared correlation (absolute correlation) with its corresponding input mask a and output mask b.

1: $counter = 0$.
2: **for** $j = 1 \rightarrow l$ **do**
3: Extract the jth Column C_j from M.
4: **repeat**
5: $C_j = M \times C_j$.
6: Increment *counter*
7: **until** *counter* equals $r - 1$.
8: tempCorr(j) = max$(|C_j|)$.
9: tempIndex(j) = The index of max$(|C_j|)$ (which gives us the corresponding input mask).
10: **end for**
11: **return** max(tempCorr) which yields the maximum average squared correlation (absolute correlation) and its index yields the corresponding output mask b. Then the corresponding input mask a = max(tempIndex(b)).

most convenient way is to consider correlation matrices with Hamming weight up to 2 or 3 bits depending on the size of the block cipher. Then, try to perform matrix-matrix multiplications and find the active input and output Sboxes that yield the maximum absolute value as they would probably be the Sboxes that yield the maximum value when considering matrices using Hamming weight more than 3. For example, experiments on the PRESENT correlation submatrix with Hamming weight up to 3, where we are able to perform matrix-matrix multiplication and thus determine the correlations of all the approximations, showed us that the best linear approximations often come from an input mask activating only one Sbox and also an output mask activating only one Sbox (which get permuted afterwards).

4.1.2 Improved Linear and Differential Approximations

We use the approach described in section 4.1.1 and report the best linear and differential approximations we found in PRESENT and SPONGENT. We use the time complexity formula, given at section 4.1.1, to specify the set of parameters ($n \equiv$ block size and $m \equiv$ maximum Hamming weight), where constructing the sparse matrices would be computationally tractable in PRESENT and SPONGENT.

As shown in Table 4.1, the number of elements in the correlation and differences submatrices

Table 4.1: $n \equiv$ Cipher's block size, $m \equiv$ maximum Hamming weight of inputs and outputs, $size \equiv$ matrix size, $nnzC \equiv$ non zero elements of the Correlation matrix, $nnzD \equiv$ non zero elements of the difference matrix, Time complexity of the correlation or difference matrix construction $\equiv O(t)$, $- \equiv$ bounded by t since each step in t fills an entry in the correlation or difference matrix. The time complexity unit is simple arithmetic operations.

Cipher	n	m	$\log_2(size)$	$\log_2(nnzC)$	$\log_2(nnzD)$	$\log_2(t)$
PRESENT	64	4	19.37×19.37	23.26	18.41	27.83
PRESENT	64	5	22.99×22.99	-	-	33.85
PRESENT	64	6	26.31×26.31	-	-	39.61
PRESENT	64	7	29.39×29.39	-	-	45.33
SPONGENT	88	4	21.22×21.22	23.63	19.18	29.58
SPONGENT	88	5	25.31×25.31	-	-	36.04
SPONGENT	88	6	29.12×29.12	-	-	42.26
SPONGENT	136	4	23.74×23.74	-	-	32.00
SPONGENT	136	5	28.48×28.48	-	-	39.02

of both PRESENT and SPONGENT is huge. A standard matrix representation would cost $2^{41.74}$ and $2^{45.44}$ bytes for the difference matrix of PRESENT and SPONGENT respectively. This is more than 1 TB. Therefore, we need to avoid running out of memory by using a sparse matrix representation which reduces memory by only allocating space for the nonzero elements. This will also speed the matrix-vector or matrix-matrix multiplications which we perform to find the best linear and difference approximations.

Table 4.1 shows us that our submatrices are very sparse, for instance the first table entry indicates that the density ($=\frac{nnz}{Size \times Size}$) of the difference transition submatrix of PRESENT with input and output differences of Hamming weight up to 4 is 7.56×10^{-7}. This confirms that these large submatrices are considerably sparse. Therefore, using a sparse matrix storage format where we only allocate storage for the nonzero elements, our large correlation submatrix could easily fit in memory. The very general and simple format for storing sparse matrices is called Compressed Column Storage (CCS). Using this format, the storage cost of a sparse matrix depends on the number of its nonzero elements (nnz) and its column size ($ncol$). More specifically, the cost of a real-valued sparse matrix in CCS format is equivalent to the cost of nnz real-valued numbers and $(nnz + ncol + 1)$ integers [102]. Thus, on a 64-bit machine, where we have 8 bytes for both real and integer numbers, the total memory cost would be $8nnz + 8(nnz + ncol + 1)$ bytes. Using the numbers on Table 4.1, we see that the total memory cost for a CCS sparse representation of PRESENT and SPONGENT difference submatrices is 11014024 ($\approx 2^{23.4}$) and 29085768 ($\approx 2^{24.8}$) bytes respectively. Also the memory cost for a CCS representation of PRESENT and SPONGENT correlation submatrices is 165990920

($\approx 2^{27.3}$) and 226974888 ($\approx 2^{27.8}$) bytes respectively. Each of these submatrices costs less than 1 GB and thus would easily fit in memory.

Application on PRESENT:

We use the approach described above to find better differential and linear approximations.

Differential Approximations: We use a difference transition submatrix whose input and output differences have Hamming weight less than or equal to four. Now, in order to estimate the differential probability after r rounds, we raise our transition submatrix to r and extract the maximum entry. The time and memory costs of this are negligible. We found that the 2-round iterative characteristic in [23] has probability 2^{-74} for 15-round PRESENT but the differential containing this characteristic has a higher probability equivalent to $2^{-63.50}$ for a 15-round PRESENT. We also found many differentials with probability larger than 2^{-64} for 16 rounds PRESENT where the maximum one occurs with probability $2^{-62.58}$.

Moreover, the maximum differential probability we found for 25 rounds is equal to $2^{-97.38}$. This is larger than the 2^{-100} differential characteristic bound for 25 rounds given in [23]. Here we note that the analysis provided in [23] is sound as the authors gave a bound for the differential characteristic and not for the differential which is hard to bound. Nevertheless, this shows that our approach can be useful in bounding the probability of a differential.

Linear Approximations and Statistical Saturation Attacks: All the previous linear attacks on PRESENT used linear trails activating only one Sbox at each round. To find better linear approximations, we considered trails activating at most 4 Sboxes. Thus, we constructed a correlation submatrix using input and output masks of Hamming weight at most 4 bits. By searching for the best approximations among input masks and output masks in one Sbox. We found that there are many approximations whose squared correlation is larger than 2^{-64} when ≤ 24 rounds of PRESENT are used.

As noted in [93], these approximations follow the normal distribution with mean zero and variance equal to their squared correlation. Thus, the squared correlation is higher for 32% of keys compared to the whole key space for some approximations where each approximation has a different path. Thus when using multiple linear approximations, each key is more likely to yield a high correlation with respect to some input and output masks [64]. Therefore statistical saturation distinguishers based on linear approximations whose squared correlations are larger than 2^{-64} work exactly as predicted for almost all the keys.

Table 4.2 lists the 10 approximations spanned from U_{11} and V_1 and also the 10 approximations spanned from U_{11} and V_3. All these approximations have a squared correlation larger than 2^{-64} and they give us two 24-round statistical saturation distinguishers. Using the input subspace

$$U_{11} = span\{e_{41}, e_{42}, e_{43}, e_{44}\}$$

which corresponds to fixing the 4 bits entering the 11-th Sbox (counting from left to right) and the output subspace

$$V_1 = span\{e_1, e_{17}, e_{33}, e_{49}\}$$

which corresponds to the 4 bits resulted after applying the permutation on the output of the first Sbox, we get a statistical saturation distinguisher on 24 rounds with an average capacity equal to $2^{-60.53}$. Using the same input subspace with another different output subspace

$$V_3 = span\{e_3, e_{19}, e_{35}, e_{51}\}.$$

We also get a statistical saturation distinguisher with an average capacity $2^{-60.53}$. Using another input subspace

$$U_{10} = \{e_{37}, e_{38}, e_{39}, e_{40}\}$$

with each of the above two output subspaces we get distinguishers with the same capacities. Now all these 24-round statistical saturation distinguishers can be used to mount a key recovery attack for 16 bits of the last round key on 25 rounds of PRESENT using the whole code book. However, using a 23-round distinguisher a key recovery attack on 25 rounds similar to the one in [38] could be mounted using less than the code book.

Moreover, these 24-round distinguishers could be used to mount a 26-round key recovery attack similar to [38] to recover 16 bits from the 1st round subkey (4 bits from each of the 9th, 10th, 11th and 12th Sbox) and also 16 bits from last round subkey (4 bits from each of the 1st, 5th, 9th and 13th Sbox) but still estimating the success probability and data complexity is difficult. However, the statistical framework developed in [63] in order to estimate the success probability and data complexity of the multidimensional attack could also be used to estimate the success probability and data complexity of this attack, should we assume the independence of the linear approximations used which is not true. This is in fact what has been done in [38] as it has been noted in [64] that the linear approximations used in the 26-round multidimensional attack of PRESENT can not be statistically independent as several approximations share the same input mask.

Therefore, rather than giving the success probability and data complexity, we list in Table 4.3 the estimated squared Euclidean distance of the statistical saturation distinguisher with input subspace U_{11} and output subspace V_1 for various number of rounds along with the

experimental Euclidean distance using 100 random master keys. We also list the Euclidean distance obtained via a wrong key guess which was simulated by encrypting one more round under the right key. Table 4.3 clearly shows that the experimental Euclidean distances are close to the estimated capacities and the more plaintext we use the closer our experimental distances get to the expected distances. Thus, using the above mentioned four statistical distinguishers we could find 16 key bits from each of the first and last round keys using the whole code book. We note that these statistical saturation distinguishers are better than the distinguisher reported in the original attack [39] whose input and output subspaces are

$$U = V = span\{e_{22}, e_{23}, e_{26}, e_{27}, e_{38}, e_{39}, e_{42}, e_{43}\}.$$

This is because all the linear approximations spanned from U and V do not have a single linear approximation with a squared correlation larger than 2^{-64} even when considering input and output masks with Hamming weight at most 4 bits.

Application on SPONGENT**:**

Here we find linear approximations that can be used to distinguish 23 rounds of the SPONGENT-88 permutation using the whole code book and this is one more round than what has been provided in [22]. We also give the maximum differential characteristic probability we found on 16-round SPONGENT-88.

Differential Approximations: We constructed a difference transition submatrix for SPONGENT-88 with input and output differences having Hamming weight at most 4 bits. The maximum differential probability obtained by powering the transition submatrix[2] is a 16-round differential and it has probability $2^{-77.83}$. One of the differentials having this probability is

$$e_1 \oplus e_4 \oplus e_{17} \oplus e_{20} \rightarrow e_9 \oplus e_{33} \oplus e_{75} \oplus e_{77}$$

and it consists of only one differential trail. This is one round less than the best differential provided by the designers as their differential include characteristics with differences having Hamming weight more than 4. It would be interesting to see whether input and output differences with Hamming weight at most 5 bits would yield better estimations. However, as noted in Table 4.1 the time complexity is 2^{36} arithmetic operations which have not tried due to the

[2]This is possible for the difference submatrices of PRESENT and SPONGENT but not for their correlation matrices as they are dense

Table 4.2: $\alpha \equiv$ input mask, $\beta \equiv$ output mask, $(C^{\leq 4}(\alpha,\beta))^2 \equiv$ squared correlation of a 24-round PRESENT linear approximation with input mask α and output mask β computed via a correlation submatrix with Hamming weight at most 4. The first 10 approximations correspond to the output subspace V_1 while the second 10 approximations correspond to the output subspace V_3. $e_i \equiv$ the unit vector with single 1 at position i whose length is 64 (PRESENT's block size). The values between parentheses represent the $\log_2$ of the corresponding number of trails which are easily calculated by replacing each nonzero entry with 1 in the correlation submatrix and then powering it to r

α	β	$\log_2((C^{\leq 4}(\alpha,\beta))^2)$
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.98 (91.67)
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.77 (90.62)
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33} + e_{49}$	-63.97 (91.48)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.80 (91.00)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{49}$	-63.97 (91.12)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.97 (91.52)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.60 (89.95)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.77 (90.47)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33} \oplus e_{49}$	-63.80 (91.48)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{17} \oplus e_{33} \oplus e_{49}$	-63.96 (91.33)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.98 (91.66)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.78 (90.62)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.97 (91.48)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.81 (91.00)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{51}$	-63.97 (91.11)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.97 (91.51)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.60 (89.95)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.77 (90.47)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.80 (90.81)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.97 (91.33)

lack of computing resources.

Linear Approximations: As mentioned in Chapter 3 (See Section 3.3.1), the SPONGENT Sbox was chosen carefully to avoid the many linear trails with one active Sbox in each round existing on PRESENT. Here we use a correlation submatrix with input and output masks of Hamming weight up to 4 to activate at most 4 Sboxes. As a result, we found many linear approximations with correlations larger than 2^{-44} for 23-round of SPONGENT-88. Thus, we improved the linear distinguishers provided by the designers one more round. Table 4.4 shows that correlations obtained from using correlation matrices with masks of Hamming weight at most 2 bits, 3 bits and 4 bits do not vary significantly and this might indicate that linear characteristics covering more than 4 active Sboxes per round do not have a significant effect

Table 4.3: The table shows the estimated Euclidean distance D together with the experimental Euclidean distance D' averaged over 100 random keys with various amount of plaintexts, namely 2^{10} plaintexts are used for $r = 2, 3$, 2^{12} for $r = 4$, 2^{17} for $r = 5, 6$, and 2^{20} for $r = 7, 8$. D'_* $\equiv$ the Euclidean distance for a wrong key guess (this was simulated by encrypting one more round under the same plaintexts and key guess used in evaluating D'). We note that $\log_2(D) = -\infty$ at $r = 2$ is a 2-round zero-correlation distinguisher [25] in PRESENT. We note that Zero-correlation distinguishers can easily be constructed in PRESENT for not more than 3 rounds since PRESENT achieves full-dependency after exactly 3 rounds.

r	2	3	4	5	6	7	8	23	24
$\log_2(D)$	$-\infty$	-8.00	-9.99	-12.81	-15.23	-17.79	-20.37	-55.52	-64.53
$\log_2(D')$	-10.07	-7.70	-9.67	-12.74	-14.32	-17.09	-19.06	-	-
$\log_2(D'_*)$	-7.69	-9.03	-11.38	-14.34	-16.19	-19.49	-19.92	-	-

in the total correlation. The table also shows that it is difficult to accurately estimate the total correlation of some linear approximations. For instance for 22 rounds, $|C^{\leq 2}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$ is smaller than $|C^{\leq 3}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$ but bigger than $|C^{\leq 4}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$. This suggests that the characteristics with Hamming weight 4 bits contributed negatively to the total correlation.

Since our results distinguish only 23 out of 45 rounds, they confirm that the SPONGENT Sbox was a good choice. As SPONGENT use the same but a wider bitwise permutation of PRESENT, we used a variant of PRESENT with the SPONGENT Sbox to check its resistance against linear attacks. We found that there are no trails with one active Sbox at each round when the number of rounds is larger than 6. Moreover, using submatrices with entries that have Hamming weight at most 4, we found that the maximum average squared correlation for 20 rounds is only $2^{-74.48}$. This 20-round average sqaured correlation is very low compared to the best average squared correlations in 24 rounds of PRESENT shown in Table 4.2. All these give us further confirmation that the SPONGENT Sbox was a good choice.

4.2 The Effect of Key Scheduling on Differential and Linear Approximations on PRESENT-Like Ciphers

In [45], Daemen and Rijmen provided a through study about the probability distributions of differential and linear approximations in block ciphers in general and especially for the so-called key alternating ciphers. Since PRESENT-like ciphers are key alternating ciphers,

Table 4.4: $r \equiv$ number of rounds, $\alpha \equiv$ input mask, $\beta \equiv$ output mask, $|C^{\leq i}| \equiv$ correlation using a submatrix with input mask α and output mask β having Hamming weight at most i bits, $- \equiv$ not applicable. $e_i \equiv$ the unit vector with single 1 at position i whose length is 88 (SPONGENT-88's block size).

r	α	β	$\log_2(\lvert C^{\leq 2}\rvert)$	$\log_2(\lvert C^{\leq 3}\rvert)$	$\log_2(\lvert C^{\leq 4}\rvert)$
22	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.82	-43.83
23	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.81	-43.74
22	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47} \oplus e_{69}$	-	-	-43.83
23	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47} \oplus e_{69}$	-	-	-43.75
22	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{51}$	-42.06	-42.05	-42.05
23	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{51}$	-44.02	-44.01	-43.95
22	$e_{70} \oplus e_{71}$	$e_{56} \oplus e_{78}$	-42.03	-42.03	-42.04
23	$e_{70} \oplus e_{71}$	$e_{56} \oplus e_{78}$	-43.99	-43.99	-43.96
22	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{29} \oplus e_{51} \oplus e_{73}$	-	-	-42.04
23	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{29} \oplus e_{51} \oplus e_{73}$	-	-	-43.94
22	$e_9 \oplus e_{10} \oplus e_{11}$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.99	-43.93
23	$e_9 \oplus e_{10} \oplus e_{11}$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.97	-43.88
22	$e_{46} \oplus e_{47} \oplus e_{48}$	$e_{34} \oplus e_{56} \oplus e_{78}$	-	-42.72	-42.69
23	$e_{46} \oplus e_{47} \oplus e_{48}$	$e_{34} \oplus e_{56} \oplus e_{78}$	-	-43.95	-43.89

then it is expected that the distribution of differential and linear approximations will be as described in [45].

However, in the following we consider the behaviour of two ciphers that use identical round keys, namely PRINTcipher and a variant of PRESENT with identical round keys. Their behaviour is questionable since their round keys are identical and thus not independent. More specifically, we study the distribution of differential approximations in PRINTcipher and this was mainly motivated by the differential attack given in Chapter 5. We also study the distribution of linear approximations in PRESENT with identical round keys to see the effect of key scheduling on the distribution of linear approximations in PRESENT.

Distribution of Differential Approximations

We are interested on the distribution of the number of right pairs of a differential approximation over all the keys. Firstly, we re-write the hypothesis of stochastic equivalence mentioned in Chapter 3 as follows: Let p_k be the probability of the differential when the secret key k is used. Then the average probability over all the key space is $\tilde{p} = \frac{1}{|K|} \sum_k p_k$. Now the *hypothesis of stochastic equivalence* suggests that $p_k \approx \tilde{p}$ [7]. Secondly, We recall the result of [45] about the distribution of the right pairs following a differential.

Proposition 4.2.1. *[45]*

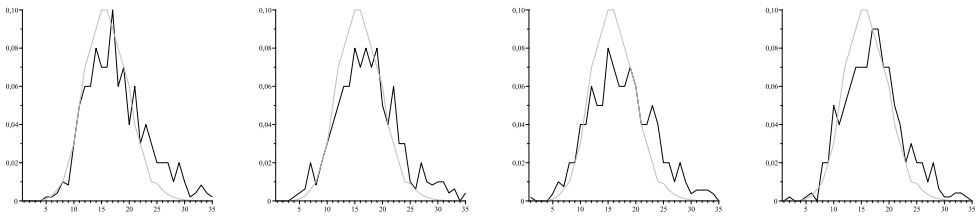
In an n-bit key alternating cipher, the number of the right pairs X following a differential with probability p is a random variable that follows the Binomial distribution $\mathcal{B}(2^{n-1}, 2^{n-1}p)$ *and also the Poisson distribution (since p is small) with mean* $\lambda = 2^{n-1}p$*:*

$$\Pr(X = i) \approx \binom{2^{n-1}}{i} p^i (1-p)^{2^{n-1}-i} \approx \frac{\mathrm{e}^{-\lambda}\lambda^i}{i!}$$

Experiments in [17] showed that the behaviour of differential approximations for a small variant of PRESENT follows the binomial distribution and they further showed that there is no significant difference between the expected probability and the mean averaged over all the keys. Thus indicating that small variants of PRESENT and probably most PRESENT-like ciphers satisfy the hypothesis of stochastic equivalence. Furthermore, recently experiments in [7] also confirmed this behaviour.

However, in PRINTcipher the distribution might be different since the probability of a differential characteristic is based on the assumption of independent round keys and PRINTcipher uses identical round keys. Therefore, we ran some tests to see if the theoretical probability $\left(\frac{1}{4}\right)^r$ of an input difference characteristic with Hamming weight one yielding an output difference characteristic with Hamming weight one is actually met (See Chapter 5). Our experimental data depicted in Figure 4.1 suggests that indeed the probability is slightly higher than expected which is due to the effect of intermediate characteristic values with Hamming weight more than one which all add to the differential probability of input and output differences with Hamming weight one.

Fig. 4.1: Experimental vs. theoretical estimates for the optimal differentials. The x-axis shows the number of pairs yielding the correct output difference within 2^{2r+4} tries. The y-axis shows the relative frequency.



(a) 9 rounds (b) 10 rounds (c) 11 rounds (d) 12 rounds

Linear Approximations' Correlation Distribution

Assuming independent round-keys, we can compute the average and variance of the correlation distribution of a linear approximation over all the keys. As the average of a sum of random variables is the sum of the averages, the average bias is zero. Here, independent round-keys are used to ensure that each single trail has average zero. Moreover, one can see that two distinct linear trails C_i and C_j are pairwise independent. As the expected value of the correlation distribution is zero, the variance of the distribution equals the average squared correlation which equals the sum of the squares of the correlations of all trails (cf. Theorem 21 in [45]). We summarize this in the following proposition.

Proposition 4.2.2. *Let $F = F_r \circ \cdots \circ F_2 \circ F_1$ be an r-round block cipher with independent round-keys, the average correlation is zero, i.e.,*

$$\mathbb{E}(C) = 0.$$

Moreover, the average squared correlation is given by

$$\mathbb{E}(C^2) = \sum_i \Pi_{j=1}^{r} c_{F_j}(\alpha_{(j-1)i}, \alpha_{ji})^2.$$

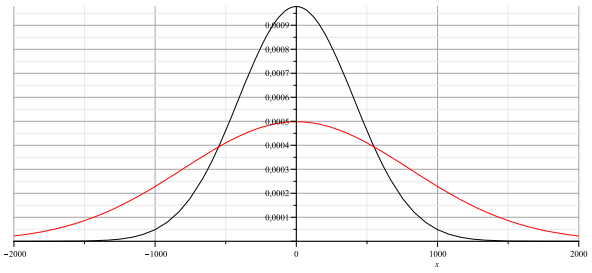
PRESENT with Independent Round-keys

As mentioned before in Chapter 3, it was experimentally confirmed in [93] that the distribution of the correlation of some linear approximations activating one Sbox per round nicely follows a normal-distribution with mean zero and variance $2^{(-2r)^2}N$ (this can be shown using the above proposition), where N is the number of the linear trails with input and output masks of Hamming weight one and only one active Sbox per round. Thus, experimentally, we can notice two facts: Firstly, for PRESENT different trails behave like independent random variables and secondly, the contribution of the non-optimal trails (those activating more than one Sbox at some rounds) does not significantly influence the distribution of approximations with input and output masks with Hamming weight one. For instance, for 20-round PRESENT using e_{43} as an input mask and e_{43} as an output mask and activating only one Sbox per round, the average squared correlation for the linear approximation (e_{43}, e_{43}) is $2^{-55.71}$. Now activating at most 4 Sboxes per round, we see no significant improvement as the average squared correlation for the linear approximation (e_{43}, e_{43}) is $2^{-55.59}$.

PRESENT with Identical Round-keys

Let us consider a variant of PRESENT with identical round-keys (and round-constants to avoid trivial slide attacks [16] described in Chapter 2). As it turns out this is an intriguing example of the influence of the key-scheduling on the distribution of the correlations. We started by performing experiments on a large number of random keys and observed that the total variance of the correlation distribution for some linear approximations of PRESENT with identical round-keys is consistently bigger than that of standard PRESENT for any number of rounds ≥ 5. Fig. 4.2 shows the distribution of the linear correlation for the identical round-keys case vs. the original PRESENT key-scheduling for 17 rounds.

Fig. 4.2: The (normalized) probability distribution of the PRESENT-cipher with the usual vs the identical round-keys case



The difference is significant in the sense that more rounds of PRESENT with identical round-keys are vulnerable to linear attacks for a non-negligible fraction of keys. In other words, the choice of the key-scheduling that makes the cipher secure or insecure against linear cryptanalysis. To illustrate the difference consider a 20 round version. The fraction of keys with a squared correlation larger than 2^{-53} is 33.7% in the case of identical round-keys but only 1.1% in the case of the standard PRESENT key-scheduling.

For computing the variance of a sum of random variables it is sufficient to study the pairwise covariance of the summands. In order to compute this variance, we need to figure out which trails are distinct and which are non-distinct. So let us, suppose that we have the following random variables, S_i and S_j, representing the signs of the correlation of the ith trail and the jth trail respectively of an r-round linear approximation (α, β) with input and output masks of Hamming weight one and only one active Sbox per round in PRESENT with identical round keys. Let $(\alpha = \alpha_{0i}, \alpha_{1i}, \cdots, \alpha_{ri} = \beta)$ denote the ith trail and $(\alpha = \alpha_{0j}, \alpha_{1j}, \cdots, \alpha_{rj} = \beta)$ denote the jth trail. Thus,

$$S_i = (-1)^{<\alpha_{0i},k> \oplus <\alpha_{1i},k> \oplus \cdots \oplus <\alpha_{ri},k>}$$

and

$$S_j = (-1)^{<\alpha_{0j},k>\oplus<\alpha_{1j},k>\oplus\cdots\oplus<\alpha_{rj},k>}$$

where k is the key at each round. One can see that S_i and S_j are identical when

$$\bigoplus_{x=0}^{r} \alpha_{xi} = \bigoplus_{x=0}^{r} \alpha_{xj}$$

In other words two trails are identical iff the (xor) sums of all intermediate masks are identical. While in general, computing the number of trails is much more efficient than listing all trails, it is still feasible for $r \leq 20$ to compute the list of trails and sort this list according to the sum of the intermediate masks. Thus, for $r \leq 20$ we can relatively easily compute the expected variance for the PRESENT-variant with identical round-keys. Table 4.5 shows the expected variance (Var_2) of the bias distribution of the optimal linear approximation for a specific one bit input and output difference (i.e. input mask $= e_{43}$, output mask $= e_{43}$). For PRESENT with identical round-keys the expected variance is very close to the observed variance (ObsVar) sampled over 20000 random keys. Table 4.5 also shows the expected variance of the bias distribution of the optimal linear approximation of (standard) PRESENT (Var_1) along with the number of trails (N_1) with one active Sbox per round, and the number of independent trails (N_2), for number of rounds r, $5 \leq r \leq 20$.

In the following, we explain exactly what happens for the distribution of the correlation of the linear approximation (e_{43}, e_{43}) of 5 rounds of PRESENT using the same round key $k = (k_1, k_2, \cdots, k_{64})$ and round constants affecting few of the rightmost bits. From Table 4.5, we see that the linear approximation with input mask and output mask equal to e_{43} has 9 trails where 5 of them are independent. As we know in PRESENT all the linear trails with one active Sbox per round have the same correlation absolute value $\left(\frac{1}{4}\right)^5$ and only differs in the sign of the correlation. The sign of the correlation of each trail represents a random variable S_i and it is a product of the sign of correlation from the Sboxes correlations and the sign of the correlation from the round keys and round constants. Now for all the trails, the sign from the Sboxes when 5 rounds are used is always positive and there are no round constants involved in all the trails. More precisely, we have the following expression for the sign of each trail

Table 4.5: Analytical and experimental data on r-round reduced PRESENT (possibly with identical round-keys). N_1 is the number of all linear trials with one active Sbox. N_2 is the number of trails (among N_1) that are independent from all the other trails. Var_1 gives the expected variance of the bias of the optimal linear approximation of (standard) PRESENT, while Var_2 corresponds to PRESENT with identical round-keys. ObsVar is the experimentally observed variance sampled over 20000 random keys.

r	N_1	N_2	$\log_2(\mathrm{Var}_1)$	$\log_2(\mathrm{Var}_2)$	$\log_2(\mathrm{ObsVar})$
5	9	5	-16.83	-16.30	-16.28
6	27	13	-19.25	-18.45	-18.47
7	72	19	-21.83	-20.93	-20.94
8	192	51	-24.42	-23.27	-23.29
9	512	101	-27.00	-25.85	-25.84
10	1344	180	-29.61	-28.40	-28.43
11	3528	318	-32.22	-30.87	-30.86
12	9261	498	-34.82	-33.23	-33.34
13	24255	723	-37.43	-35.74	-35.73
14	63525	991	-40.04	-38.18	-38.34
15	166375	1184	-42.66	-40.71	-40.71
16	435600	1232	-45.26	-43.15	-43.28
17	1140480	1143	-47.88	-45.63	-45.61
18	2985984	890	-50.49	-48.03	-48.12
19	7817472	575	-53.10	-50.50	-50.52
20	20466576	300	-55.71	-52.88	-52.92

$$S_1 = (-1)^{k_{43} \oplus k_{43} \oplus k_{43} \oplus k_{43} \oplus k_{43} \oplus k_{43}}$$
$$S_2 = (-1)^{k_{43} \oplus k_{43} \oplus k_{27} \oplus k_{39} \oplus k_{42} \oplus k_{43}}$$
$$S_3 = (-1)^{k_{43} \oplus k_{43} \oplus k_{11} \oplus k_{35} \oplus k_{41} \oplus k_{43}}$$
$$S_4 = (-1)^{k_{43} \oplus k_{27} \oplus k_{39} \oplus k_{42} \oplus k_{43} \oplus k_{43}}$$
$$S_5 = (-1)^{k_{43} \oplus k_{27} \oplus k_{23} \oplus k_{38} \oplus k_{42} \oplus k_{43}}$$
$$S_6 = (-1)^{k_{43} \oplus k_{27} \oplus k_{7} \oplus k_{34} \oplus k_{41} \oplus k_{43}}$$
$$S_7 = (-1)^{k_{43} \oplus k_{11} \oplus k_{35} \oplus k_{41} \oplus k_{43} \oplus k_{43}}$$
$$S_8 = (-1)^{k_{43} \oplus k_{11} \oplus k_{19} \oplus k_{37} \oplus k_{42} \oplus k_{43}}$$
$$S_9 = (-1)^{k_{43} \oplus k_{11} \oplus k_{3} \oplus k_{33} \oplus k_{41} \oplus k_{43}}$$

From the above equations, we see that

$$S_2 = S_4 = (-1)^{k_{27} \oplus k_{39} \oplus k_{42} \oplus k_{43}}$$

and

$$S_3 = S_7 = (-1)^{k_{11} \oplus k_{35} \oplus k_{41} \oplus k_{43}}$$

and the other five random variables S_1, S_5, S_6, S_8 and S_9 are independent from them. Now the correlation of each trail $C_i = (\frac{1}{4})^5 S_i$ is a random variable following the binary distribution with mean zero and variance $C_i^2 = (\frac{1}{4})^{10}$. Thus, the variance of the distribution of the linear approximation (e_{43}, e_{43}), Var_2 is calculated as follows:

$$\begin{aligned}
\mathrm{Var}_2 &= \mathrm{Var}(C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + C_9) \\
&= \mathrm{Var}(C_1 + 2C_2 + 2C_3 + C_5 + C_6 + C_8 + C_9) \\
&= \mathrm{Var}(C_1) + 4\mathrm{Var}(C_2) + 4\mathrm{Var}(C_3) + \mathrm{Var}(C_5) + \mathrm{Var}(C_6) + \mathrm{Var}(C_8) + \mathrm{Var}(C_9) \\
&= 13C_i^2 \approx 2^{-16.30}.
\end{aligned}$$

4.3 Conclusion and Future Work

In this chapter, we used sparse difference and correlation submatrices to estimate the probabilities of low-weight differential and linear approximations respectively in PRESENT-like ciphers. This estimation approach can also be used in any cipher allowing low-weight differential and linear characteristics. Using these sparse matrices, we found linear distinguishers for 23-round of SPONGENT-88. While this is far from distinguishing the full 45 rounds of SPONGENT-88, it is the best currently known result against SPONGENT. We also presented four 24-round statistical saturation distinguishers which break 26-round of PRESENT and that is more than the rounds attacked by the original statistical saturation attack [39].

It would be interesting to investigate whether using large difference and correlation submatrices for PRESENT and SPONGENT-88 with entries having Hamming weight at most 5 would make some improvements over this work. Looking at Table 4.1 we see that the time complexities for constructing these submatrices take around 2^{34} and 2^{36} arithmetic operations for PRESENT and SPONGENT-88 respectively which could be feasible using parallel computing.

We also showed that the key scheduling algorithm can significantly affect the distribution of linear approximations in PRESENT-like ciphers (See also Chapter 6). More specifically, we showed that the distribution of linear approximations with input and output masks of

Hamming weight one in PRESENT with identical round keys is significantly larger than PRESENT with independent round keys. As a further work, it would be interesting to compare the correlation distribution in PRESENT with identical round keys and PRESENT with independent round keys when input and output masks of Hamming weight more than one are used since these input and output masks yield better approximations as shown in Table 4.2.

Chapter 5

Differential Cryptanalysis of Round-Reduced PRINTcipher: Computing Roots of Permutations

In this chapter, we mount a differential attack on the block cipher PRINTcipher described in Chapter 3 (See Section 3.1.2). As PRINTcipher uses a key-dependent permutations, it might seem first that mounting a differential attack on PRINTcipher is difficult due to the unknown bit permutations which make it impractical to use the submatrix approach method explained in the previous Chapter. We show in this chapter that this is not the case. We present two differential attacks that successfully break about half of the rounds of PRINTcipher.

Moreover, one of the attacks is of independent interest, since it uses a mechanism to compute roots of permutations. If an attacker knows the many-round permutation π^r, the algorithm can be used to compute the underlying single-round permutation π. This technique is thus relevant for all iterative ciphers that deploy key-dependent permutations. In the case of PRINTcipher, it can be used to show that the linear layer adds little to the security against differential attacks.

We close the chapter by explaining how to design an authentication protocol whose security is based on a multi-valued function, namely, the r-th roots of a permutation in the symmetric group S_n.

5.1 Using Differential Cryptanalysis To Recover the Permutation Key

For PRINTcipher, the classical differential attacks described in Chapter 2 can not be directly applied in a straightforward fashion, since finding good differentials requires the knowledge of the linear layer, which for PRINTcipher is key-dependent and thus unknown. As already pointed out, however, this disadvantage can also be turned into an advantage for the attacker:

It can be used to learn something about the part of the key that defines the linear layer.

5.1.1 Optimal Differential Characteristic

We start our analysis by proving the following fact about the optimal PRINTCIPHER characteristic.

Theorem 5.1.1. *Given an input difference α of weight one, the unique most probable r-round differential characteristic is*

$$\alpha \to (P \circ K)(\alpha) \to (P \circ K)^2(\alpha) \to (P \circ K)^r(\alpha),$$

which will occur with probability $\left(\frac{1}{4}\right)^r$, where the symbol '$\circ$' refers to the composition operation of the symmetric group (S_{48} or S_{96}).

Proof. The difference distribution table for the PRINTCIPHER Sbox (See Table 3.4) shows that all occurring differences are equally probable (prob. $\frac{1}{4}$) and that for every 1-bit input difference, there exists exactly one 1-bit output difference. From this, it follows that starting with a 1-bit input difference, a 1-bit differential trail through r rounds of PRINTCIPHER occurs with probability $\left(\frac{1}{4}\right)^r$. Note also that this trail has the minimum possible number of r active Sboxes and that no other Sbox difference is more probable, meaning that this trail is the most probable one.

Also note that in Table 3.4 the 1-bit output difference always occurs in the same bit position as the 1-bit input difference. This means that if the 1-bit differential occurs, the Sbox does not permute the active bit - its position on the differential trail is only influenced by the fixed permutation P and the key-dependent permutation K. Thus, the difference α is indeed mapped to $(P \circ K)^r(\alpha)$, which proves the theorem. □

5.1.2 Targeting the Xor Key

In the following, we assume that the attacker has the full code book at his disposal (i.e. 2^{48} plaintext/ciphertext pairs for r rounds of PRINTCIPHER-48). For every 1-bit input difference $\alpha_1 = (100...0), \alpha_2 = (010...0), \ldots, \alpha_{48} = (000...1)$, the attacker now forms all 2^{47} input pairs with $x \oplus x' = \alpha_i$ and checks whether the output difference also has weight one. If yes, he assumes that he has found the above optimal characteristic. It turns out that as long as

$r \leq 22$, this is very likely to happen[1].

Every successful 1-bit differential gives the attacker information about the internal behaviour of the cipher which can be used to reconstruct part of the xor key. Consider the first round of the cipher and note that according to [77], the order of Sbox and key-dependent permutation can be inversed by adding two constants c and d that do not affect the differential. Thus, we can alternatively consider one PRINTCIPHER round to consist of key addition, fixed permutation, round constant, adding c, Sbox, key-dependent permutation, and adding d. In particular, for the purposes of differential cryptanalysis, we can assume the Sbox to follow directly after the key addition.

Now consider a successful differential with input difference $\alpha_1 = (100...0)$. Three key bits (with indices 1, 17 and 33) will affect the bits that go into the first Sbox. There are *a priori* 8 possible choices for these bits, generating all possible 3-bit Sbox input pairs with difference α_1. However, as shown in Table 3.4, only 2 of them will lead to a 1-bit output difference after running through the Sbox. Thus, only $\frac{1}{4}$ of all keys meet the condition for the first Sbox, reducing the key entropy by 2 bit. Thus, finding 16 successful 1-bit to 1-bit differentials (one for each Sbox) will reduce the key entropy by 32 bit, leaving a brute-force effort of 2^{48} steps. This work factor could be reduced further, but without greatly affecting the overall running time, which is dominated by the 2^{48} steps of computing the full code book anyway.

The False Positive Problem: The above description is a simplification since it does not take false positives into account. For every 1-bit differential, trying out 2^{47} plaintext pairs will yield $2^{47} \cdot \frac{48}{2^{48}} = 24$ false positives on average, i.e. 1-bit output differences that occur accidentially and not as a result of the correct differential. The question remains how they can be distinguished from the cases where the 1-bit output differences really result from the desired differential. It turns out that for 22 rounds, the probability that all 48 differentials are met at least three times is 0.514, meaning that in more than half of the cases, the correct 1-bit difference should be recognizable by occuring more often than the false positives, which very rarely occur more than twice.

5.1.3 Targeting the Linear Layer

As it turns out, there is also a different way of using the above differential to cryptanalyze PRINTCIPHER. Remembering that according to Theorem 5.1.1, *every* 1-bit to 1-bit char-

[1] We have 2^{47} pairs and a success probability of $\left(\frac{1}{4}\right)^{22} = 2^{-44}$, yielding a success probability close to 1 for any single index i and of ≈ 0.984 for all 48 indices. When increasing the number of rounds to $r = 23$, the success probability drops to 0.865 for any single index and to 0.001 for all 48 indices.

acteristic is optimal and describes the mapping $\alpha \rightarrow (P \circ K)^r(\alpha)$, the following corollary immediately follows:

Corollary 5.1.2. *Learning all optimal characteristics is the same as learning* $(P \circ K)^r$.

If the attacker has the full code book available, he can form 2^{47} plaintext pairs for every 1-bit input difference. The probability that at least one example of all 48 1-bit differentials is found is 0.984, and as stated above, the probability that they all can be distinguished successfully from false positives is 0.514. Thus, for up to $r = 22$ rounds of PRINTCIPHER-48, the attacker can learn the permutation $(P \circ K)^r$.

If he can find the r-th root of this permutation, then he has derived $P \circ K$ and thus the linear layer key K. Once this has been done, the xor key can be retrieved bitwise, using a simple divide-and-conquer attack similar to the one described in Subsection 5.1.2. It turns out that here too, the overall running time is dominated by computing the code book, i.e. the attack requires about 2^{48} computational steps.

This type of differential attack is the dual to the one targeting the xor key and is relevant for all SPN-like ciphers that use key-dependent permutations. For this reason, it is not only interesting for the analysis of PRINTCIPHER, but also for the understanding of key-dependent permutations in general. In the rest of this chapter, we will thus discuss the computation of permutation roots in more detail.

5.2 Finding (PRINTCIPHER)-Roots of a Permutation

From the previous section, we see that our problem of finding the permutation key can be reduced to the problem of finding the r-th roots of a given permutation in the symmetric groups, S_{48} and S_{96}, where r is the number of rounds.

Any permutation can be expressed as a product of disjoint cycles, and it is this representation that is most useful when computing roots. In particular, the permutation found through differential cryptanalysis can be expressed as a product of disjoint cycles in S_{48} and S_{96}.

As the elements of S_b ranges from 1 to b, we re-arrange the bitwise permutation of PRINTCIPHER with block size b described in Chapter 3 to be in the range from 1 to b rather than from 0 to $b-1$. The PRINTCIPHER bit permutation $P(i)$ can be re-written as follows

$$P(i) = \begin{cases} 3i - 2 \bmod b - 1 & \text{for } 1 \leq i \leq b-1, \\ b & \text{for } i = b, \end{cases}$$

where $b \in \{48, 96\}$ is the block size.

Before describing how to find a root for a permutation in general, we outline the basic ideas. For this, let us first see what happens when we raise a single cycle to the r-th power.

Let $c = (c_0, c_1, \ldots, c_{l-1})$ be a cycle of length l in S_n. Then c^2 will remain a single cycle when l is odd, namely, $c^2 = (c_0, c_2, \ldots, c_{l-1}, c_1, c_3, \ldots, c_{l-2})$, and will be decomposed into 2 cycles when l is even, namely, $c^2 = (c_0, c_2, \ldots, c_{l-2})(c_1, c_3, \ldots, c_{l-1})$. In general, depending on l, c^r will either remain a single cycle or be decomposed into a number of cycles having the same length (See Lemma 5.2.1). Each element c_i will be in a cycle, say, $(c_i, c_{i+r}, c_{i+2r}, \ldots, c_{i+(k-1)r})$, where $i + kr \equiv i \pmod{l}$ and $i + jr$ is reduced modulo l for each j. So in order to find the r-th root we have two cases, the first one is when c^r is a single cycle, and here c^r equals exactly $(c_0, c_r, c_{2r}, \ldots, c_{(l-1)r})$. The second case is when c^r consists of a number of disjoint cycles, and here we combine these disjoint cycles into a single cycle in a certain way in order to get c (see the proof of Theorem 5.2.2). To illustrate this, let us find the square root of the permutation $\sigma^2 = (1, 3, 2)(4, 6, 7)(5)(8)$ in S_8. According to the above explanation, we know that cycles of the same length are either a decomposition of a single cycle in the root σ or a reordering of a single cycle in the root σ. Considering cycles of length 1, (5) and (8), it is obvious that they arise from either (5)(8) or $(5, 8)$.

Thus, there are two possibilities for cycles of length 1 in σ. Cycles of length 3, $(1, 3, 2)$ and $(4, 6, 7)$, are either a decomposition of a single cycle in σ, this could be $(1, 4, 3, 6, 2, 7)$, $(1, 6, 3, 7, 2, 4)$ or $(1, 7, 3, 4, 2, 6)$; or a reordering of disjoint cycles in σ and this could only be $(1, 2, 3)(4, 7, 6)$. Summarizing, there are four possibilities for cycles of length 3 in σ. So the total number of square roots for the permutation σ^2 is 8.

5.2.1 The General Case

The procedure for constructing an r-th root for a permutation, described in [110], is based on the following basic fact in the theory of symmetric groups which can be easily deduced from the previous explanation.

Lemma 5.2.1. *Let $C \in S_n$ be a cycle of length l and let r be a positive integer. Then C^r consists of $gcd(l, r)$ disjoint cycles, each of length $\frac{l}{gcd(l,r)}$.*

The following theorem is due to A. Knopfmacher and R. Warlimont [110, p. 148]. We recall its proof, as the proof describes how to construct an r-th root. Throughout the rest of this chapter, we use the notation l-cycle to mean a cycle of length l.

Theorem 5.2.2. *[8, 110] Let $r = p_1^{i_1} p_2^{i_2} \dots p_n^{i_n}$, where $p_1, p_2, \dots, p_n$ are the prime factors of r. A permutation $Q \in S_n$ has an r-th root, iff for every integer $l \geq 1$, the number of l-cycles in Q is divisible by $((l, r)) := \prod_{\{j:p_j|l\}} p_j^{i_j}$.*

Proof. ($\Leftarrow$): to prove this, we construct an r-th root, R, of Q. Let a_l be the number of l-cycles in Q. Let $g = ((l, r))$. Then $a_l = gm$, where m is an integer, so we can divide the l-cycles of Q into m groups where each group consists of g l-cycles. Assume that we have the cycles, $c_{ij} = (c_{ij}^{(0)}, c_{ij}^{(1)}, \dots, c_{ij}^{(l-1)})$ where $1 \leq i \leq g$ and $1 \leq j \leq m$. For each j, we construct a cycle of length gl, say

$$R_j = (c_{1j}^{(0)}, c_{2j}^{(0)}, \dots, c_{gj}^{(0)}, c_{1j}^{(d)}, c_{2j}^{(d)}, \dots, c_{gj}^{(d)}, \dots, c_{1j}^{((l-1)d)}, c_{2j}^{((l-1)d)}, \dots, c_{gj}^{((l-1)d)}),$$

where $d = \frac{r}{g}$ and sd is reduced modulo l for each $1 \leq s \leq l-1$. Now R_j is a cycle of length gl, so according to the previous lemma, R_j^r consists of $gcd(gl, r)$ cycles of length $\frac{gl}{gcd(gl,r)}$. Now, since $g = ((l, r))$, then $gcd(l, \frac{r}{g}) = 1$ and so $gcd(gl, r) = g$, which means that R_j^r consists of g cycles of length l, namely, $c_{1j}, c_{2j}, \dots, c_{gj}$. So $\prod_{j:1\leq j\leq m} R_j$ is an r-th root for the l-cycles of Q. Repeating the same procedure for all l will yield an r-th root of Q. For the proof of ($\Rightarrow$), see [8].

□

In [55, 84], a procedure to find all the roots of Q is described. Going back to the previous theorem, we see that the main property that enables us to construct an r-th root for the l-cycles of Q is having $gcd(gl, r) = g$. Repeating the same procedure for all the g's that satisfy $gcd(gl, r) = g$ will allow us to find all the possible roots that can come from the l-cycles. Note that g is bounded by a_l (the number of l-cycles). To find all the roots, for each group consisting of l-cycles in Q, we proceed as follows.

First we construct the set $G_r(l, a_l) = \{g_i : gcd(g_i l, r) = g_i \text{ and } 1 \leq g_i \leq a_l\}$. Now, this tells us that the roots have cycles of length $g_i l$, but we do not know how many of them. For this, we solve the following Frobenius equation for $x_i \geq 0$:

$$g_1 x_1 + g_2 x_2 + \dots + g_k x_k = a_l \qquad \text{where } k = |G| \tag{5.1}$$

This equation will usually have more than one solution. Each solution corresponds to a possible cycle structure of the roots. For instance, the solution $x = (x_1, x_2, \dots, x_k)$, tells us that each corresponding root for the l-cycles of Q consists of x_i cycles of length $g_i l$ for $1 \leq i \leq k$.

The efficiency of computing all roots is of course bounded by the total number of roots. If a permutation has a huge number of roots, computing all of them is very time consuming. It is

therefore of interest to know the number of roots in advance.

In [84], using the above information about the cycle structure of permutations that have an r-th root, the following explicit formula[2] for calculating the number of all the possible roots is provided.

Theorem 5.2.3. *[84] Let r be a positive integer and $Q \in S_n$. Let a_l be the number of l-cycles in Q, where $1 \leq l \leq n$. Let $X(l, a_l)$ be the set of all the possible solutions of equation (5.1). Then the number of r-th roots of Q is*

$$\prod_{a_l \neq 0} a_l! \Big(\sum_{x \in X(l,a_l)} \prod_{i=1}^{k} \frac{l^{(g_i-1)x_i}}{g_i^{x_i} x_i!} \Big) \tag{5.2}$$

where $x = (x_1, x_2, \ldots, x_k)$ and $\{g_i : 1 \leq i \leq k\}$ are the elements of $G_r(l, a_l)$.

To get a feeling of how many roots of a permutation can be expected for the case of PRINTCIPHER-48, let us take the following permutation in S_{48}, suppose we have

$$\begin{aligned} \tau^{24} =& (1,7,47)(2,19,45)(3,48,17)(4,9,38)(5,16)(6,33,32)(8,28)(10,35)(11, \\ & 27,18)(12,20)(14,19,41)(15,46)(21,26,30)(22,34)(23,36)(25,42,39) \\ & (40,44)(13)(24)(31)(37)(43) \end{aligned} \tag{5.3}$$

So we have $a_1 = 5, a_2 = 8, a_3 = 9$ and $a_l = 0$ for $4 \leq l \leq 48$. $G(1, a_1) = \{1, 2, 3, 4\}$, $X(1, a_1) = \{(0, 1, 1, 0), (1, 0, 0, 1), (1, 2, 0, 0), (2, 0, 1, 0), (3, 1, 0, 0), (5, 0, 0, 0)\}$, $G(2, a_2) = \{8\}$, $X(2, a_2) = \{(1)\}$ and $G(3, a_3) = \{3, 6\}$, $X(3, a_3) = \{(3, 0), (1, 1)\}$. Plugging these values into equation (5.2), we find that the number of roots is $\simeq 2^{51.3}$. Moreover, the case where τ^{22} is the *Identity* has $\simeq 2^{192}$ roots in S_{48}.

Note that out of all 48!(96!) permutations only a tiny fraction of $2^{32}(2^{64})$ permutations actually correspond to a valid key in PRINTCIPHER-48(96). We can therefore expect that in the above example out of the $\simeq 2^{51.3}$ only a very small number will actually correspond to a PRINTCIPHER-permutation. In particular there is only one root for equation (5.3) that corresponds to a PRINTCIPHER permutation.

The main purpose of the next section is to describe a method that filters out wrong candidates as soon as possible, allowing to considerably speed up the computation of all valid PRINTCIPHER-roots.

[2] A more complicated formula was previously found by Pavlov in [96].

5.2.2 PRINTcipher-Roots

As discussed in the last section, computing all the roots of $(P \circ K)^r$ in order to find the right permutation key is inefficient. In this section we describe a method that finds the permutation roots $P \circ K$ belonging to the $2^{32}(2^{64})$ possible permutations in PRINTcipher-48(96). Throughout the rest of this chapter, we only discuss PRINTcipher-48 and unless mentioned explicitly, the assumption is that everything about PRINTcipher-48 follows for PRINTcipher-96 with a slight modification.

Our method uses the fact that when we apply the fixed permutation, P, for all $1 \leq i \leq 16$, the 3 bits $i, i+16$ and $i+32$ go to the ith Sbox, where depending on the permutation key, they are permuted to only four out of the six possible permutations. So the result of applying the fixed permutation, P, and then applying the keyed permutation, K, on a 48 bits plain text, is a permutation $P \circ K$ that satisfies the following two properties:

1. *Property* 1: For all $1 \leq i \leq 48$, $P \circ K(i)$ equals one of the following three possible values depending on K,

$$P \circ K(i) = \begin{cases} 3i - 2 \pmod{48} & \text{if } 3i - 2 \neq 48) \\ 3i - 1 \pmod{48} & \text{if } 3i - 1 \neq 48) \\ 3i \pmod{48} & \text{if } 3i \neq 48) \end{cases}$$

2. *Property* 2: Only 4 out of the 6 possible 3-bit permutations are valid, namely, $P \circ K(i)$, $P \circ K(i+16)$ and $P \circ K(i+32)$ are permuted to one of the four possible permutations, i.e., for all $1 \leq i \leq 48$, the following two permutations are not allowed:

   (a) $P \circ K(i) = 3i - 1$, $P \circ K(i+16) = 3i$ and $P \circ K(i+32) = 3i - 2$.

   (b) $P \circ K(i) = 3i$, $P \circ K(i+16) = 3i - 2$ and $P \circ K(i+32) = 3i - 1$.

Definition 4. *A* PRINTcipher *permutation root is any permutation on 48 elements satisfying both Property 1 and Property 2.*

Definition 5. *A* PRINTcipher *permutation(cycle) is any permutation(cycle) on less than 48 elements satisfying both Property 1 and Property 2.*

To explain these definitions, consider the following two cycles $(14, 41, 25, 26, 30, 42, 29, 39, 21)$ and $(14, 42, 30, 41, 25, 26, 29, 39, 21)$. We want to investigate whether these cycles are PRINTcipher cycles or not. The latter cycle satisfies the two properties and so it is a PRINTcipher cycle, in other words it can be part of a PRINTcipher permutation root, $P \circ K$. The former cycle satisfies only *Property* 1 but not *Property* 2 since we have $P \circ K(14) = 41$ and

$P \circ K(30) = 42$ and therefore $P \circ K(42) = 40$, and this is one of the two disallowed permutations (see item (a) in *Property* 2) and so it cannot be part of a valid PRINTcipher permutation root, $P \circ K$. Sometimes we can have a permutation consisting of two or more cycles having same or different lengths, that satisfies *Property* 1 but not *Property* 2. For example, the following permutation, $(1, 2, 6, 17)(5, 15, 44, 34)$, satisfies *Property* 1 but not *Property* 2 as we have $P \circ K(2) = 6$ and $P \circ K(34) = 5$ and therefore $P \circ K(18) = 4$, which is an invalid PRINTcipher permutation (see item (b) in *Property* 2).

Since the same cycles and permutations can be written in different ways, our method adopts the notion that starts writing each cycle by its smallest element and lexicographically order the disjoint cycles of the same length of a permutation in order to avoid repetitions in the permutation roots of $(P \circ K)^r$. Our method consists of two algorithms: the first one constructs a PRINTcipher cycle of length gl and the second one uses the first algorithm to construct k combined disjoint cycles, each of length gl. In what follows, we shall give a detailed description of the two algorithms and end this section by showing how to use Algorithm 7 to find the whole PRINTcipher permutation roots of $(P \circ K)^r$.

Finding single PRINTcipher cycles

Given a_l cycles of length l, the following algorithm constructs all the possible PRINTcipher cycles of length gl beginning with an element called *first* specified in the input (must be one the the first elements in one of these a_l cycles). The algorithm performs a depth first search to find all the other possible $g - 1$ cycles with minimal elements larger than *first* and can be combined with the cycle containing *first* as described in Theorem 5.2.2 in order to form a PRINTcipher cycle (or just reorder the given cycle in the case $g = 1$ as described previously).

Plugging all the 2-cycles of equation (5.3) and setting *first* $= 5$ and $g = 8$ will produce the following PRINTcipher cycle of length 16, $(5, 15, 44, 36, 12, 35, 8, 22, 16, 46, 40, 23, 20, 10, 28, 34)$. Algorithm 6 enables us to find a PRINTcipher permutation consisting of only one cycle of length gl but note that some of the x_i's in equation (5.1) can be more than 1. So we need another algorithm which can find a PRINTcipher permutation consisting of k disjoint cycles where $k \geq 1$.

Finding k combined PRINTcipher cycles

Given a_l cycles of length l, the following algorithm constructs a permutation beginning with an element called *first* specified in the input (must be the first element in one of these a_l cycles) and consisting of k combined and disjoint cycles ordered lexicographically. It basically

Algorithm 6 finds a PRINTCIPHER cycle of length gl
find-cycle(cycle, current, g, l-cycles)

Require: l-cycles numbered from 1 to a_l where $a_l \geq g$
Require: current = *first*, *cycle* = *first*
Ensure: *cycle* is a PRINTCIPHER cycle of length gl

```
1: for count=0 to 2 do
2:    next = 3 × current - count
3:    if next ∈ l-cycles then
4:       if next > first and next.cycleno ≠ first.cycleno and cycle.length < g then
5:          if next.cycleno ≠ the cycleno of all the elements of cycle then
6:             Add next to cycle
7:             current = next
8:             Perform again this algorithm on cycle, find-cycle(cycle, current, g, l-cycles)
9:          end if
10:      else if cycle.length = g and next.cycleno = first.cycleno then
11:         Complete the construction of cycle by combining the g different cycles to get a
            single cycle of length gl as shown in the proof of Theorem 1 (when g = 1, reorder
            the cycle containing first as described previously and assign it to cycle)
12:         if cycle satisfies Property 2 then
13:            cycle is a PRINTCIPHER cycle of length gl
14:         end if
15:      end if
16:   end if
17: end for
```

performs a recursive depth first search. The recursive algorithm begins by invoking Algorithm 6 which outputs single cycles of length gl beginning with *first*. It then proceeds from each cycle found by Algorithm 6 and concatenates it with the previously $i-1$ concatenated disjoint cycles found after the ith recursive call and if the concatenation satisfies *Property* 2, it recursively calls itself a number of times, each time with a different *first* element to begin the required permutation with as this will enable us to find all the possible $i+1$ disjoint cycles, on a reduced number of l-cycles (exactly $a_l - gi$ cycles) consisting of all the l-cycles except the gi cycles involved on the i concatenated disjoint cycles (in each invocation *first* is set to the smallest element on one of the currently available $a_l - gi$ cycles). Each recursive call stops when $i = k$, or when Algorithm 6 returns nothing, or when each concatenation of i cycles does not satisfy *Property* 2.

Using Algorithm 6 for all the possible g's along with all the possible *first* values and setting $a_1 = 48$, we can find all the possible PRINTCIPHER cycles. For instance, when $g = 1$ and $a_1 = 48$, Algorithm 6 returns four 1-cycles when trying all the possible values for *first*, namely, (1), (24), (25) and (48). When $g = 2$ and $a_2 = 48$, we found that there are six possible 2-cycles, namely, $(6, 18)$, $(7, 19)$, $(12, 36)$, $(13, 37)$, $(30, 42)$ and $(31, 43)$. When $g = 3$, we found there are eight possible 3-cycles. This information enables us to reduce the size of the cycle structure of the roots by removing any structure containing more than four 1-cycles, six 2-cycles and eight 3-cycles. It also enables us to easily find some roots, for example, knowing all PRINTCIPHER cycles of length 1 and 2, we can easily find that $(24)(13, 37)(30, 42)$ is a PRINTCIPHER permutation that is a root for the 1-cycles of equation (5.3).

Moreover, using Algorithm 7 we find that we cannot have a permutation consisting of more than 6 disjoint cycles of length 5 in PRINTCIPHER-48 and not more than 9 cycles of length 4, 12 cycles of length 5, 13 cycles of length 6 and 12 cycles of length 7 in PRINTCIPHER-96. This will generally reduce the number of solutions of equation (5.1) and therefore the size of the cycle structure which will speed up the process of finding PRINTCIPHER permutations roots.

Finding PRINTCIPHER permutations

Now, when given a_l cycles of length l, Algorithm 7 enables us to find PRINTCIPHER permutations beginning with a specified element and consisting of k cycles, each of length gl. But in order to find the rth permutation roots for all the l-cycles we use Algorithm 7 together with the elements of the sets $G(l, a_l)$ and $X(l, a_l)$. Each entry $x_j = (x_{j1}, x_{j2}, \ldots, x_{jk}) \in X(l, a_l)$ where $k = |G(l, a_l)|$, represents the cycle structure of many rth roots for the l-cycles and it might correspond to few or none PRINTCIPHER permutations, so for each $x_j \in X(l, a_l)$, we try to find all the possible PRINTCIPHER permutations beginning with a specific element

Algorithm 7 finds a PRINTCIPHER permutation that has k disjoint gl-cycles
find-k-cycles(C, current, k, g, l-cycles)

Require: l-cycles numbered from 1 to a_l where $a_l \geq g$
Require: current = *first*
Require: $C = \{\}$
Ensure: k disjoint PRINTCIPHER gl-cycles, or return $\{\}$ if there is no k disjoint PRINTCIPHER cycles

```
1: Invoke Alg. 6 on the current l-cycles
2: if number of cycles found by Alg. 1 > 0 then
3:    if the number of disjoint cycles in C consists of k − 1 disjoint cycles then
4:       for each permutation cycle found by Alg. 6 do
5:          C = C ∪ cycle
6:          if C satisfies Property 2 then
7:             return C
8:          else
9:             return {}
10:         end if
11:      end for
12:   else
13:      for each cycle found by Alg. 6 do
14:         C = C ∪ cycle
15:         if C satisfies Property 2 then
16:            Delete all the l-cycles involved in C from the a_l cycles of length l
17:            for each cycle ∈ currently available l-cycles do
18:               {Perform again this algorithm on the current l-cycles to find the other k − 1
                  cycles}
19:               current = first element in cycle
20:               find-k-cycles(C, current, k, g, l-cycles)
21:            end for
22:         end if
23:      end for
24:   end if
25: else
26:   return
27: end if
```

called *first* (must be the first element in one of these a_l cycles) and that can be roots for the l-cycles by applying Algorithm 7 through all the nonzero entries of x_j. Trying all the possible values for *first* gives us all PRINTCIPHER permutations that are roots for all the l-cycles. Now, assume that we find all the possible PRINTCIPHER permutations for each l, say σ_{l_i}, for $1 \leq i \leq \eta_l$ where η_l is the number of permutation roots of the l-cycles of $(P \circ K)^r$, so all the possible products $\prod_{a_l>0} \sigma_{l_i}$ where $1 \leq l \leq 48$ and $1 \leq i \leq \eta_l$, represent the PRINTCIPHER permutation roots which are the possible values for $P \circ K$ and by brute forcing these $P \circ K$ values we can recover the permutation key, K.

Let us try to find PRINTCIPHER permutations that are roots for the nine 3-cycles in equation (5.3). We have $G(3, a_3) = \{3, 6\}$ and $X(3, a_3) = \{(3, 0), (1, 1)\}$. We start with, $x_1 = (3, 0)$, here we only need to apply Algorithm 7 using any possible *first* because the 3 disjoint cycles of length 9 would come from all the 9 cycles. Setting *first* $= 1$ and applying Algorithm 7 doesn't give us 3 disjoint cycles of length 9, so we conclude that there is no root having the cycle structure x_1. So we go to the next cycle structure, $x_2 = (1, 1)$, we start with $x_{21} = 1$ and use Algorithm 7 on all the possible *first* values. Setting *first* $= 1, 2, 3, 4, 6$ and 11 doesn't yield a single cycle of length 9, while *first* $= 14$ yields the cycle $(14, 42, 30, 41, 25, 26, 29, 39, 21)$, we save it and continue to the next element $x_{22} = 1$ where we use Algorithm 7 on the 6 cycles that are not involved in the previous found cycle. Now we want to construct a cycle of length 18, so all the 6 cycles would be involved in it, setting *first* $= 1$, yields $(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)$. Concatenating this cycle with the previous found cycle, we get $(14, 42, 30, 41, 25, 26, 29, 39, 21)(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)$ which satisfies *Property* 2. This means that it is a PRINTCIPHER permutation that is a root for all the 3-cycles in equation (5.3). Now, we have found the roots for all the l-cycles in equation (5.3). Concatenating them together gives us the following PRINTCIPHER permutation root: $(1, 2, 4, 11, 33, 3, 7, 19, 9, 27, 32, 48, 47, 45, 38, 18, 6, 17)(5, 15, 44, 36, 12, 35, 8, 22, 16, 46, 40, 23, 20, 10, 28, 34)(14, 42, 30, 41, 25, 26, 29, 39, 21)(13, 37)(30, 42)(24)$.

5.3 Experimental Verifications

To demonstrate the efficiency of our attack we implemented the above algorithms. Experiments show that $(P \circ K)^r$ could yield more than one PRINTCIPHER root when $(P \circ K)^r$ contains several 1-cycles, but in most cases there was exactly one PRINTCIPHER root.

To derive bounds for the number of PRINTCIPHER permutations roots, we computed the number of all PRINTCIPHER permutation roots for $(P \circ K)^r = \mathit{Identity}$ where $2 \leq r \leq 22$. This seems the worst case that could happen for any r since $a_1 = 48$, which is a_1's largest value, and as shown in Table 5.1, the number of PRINTCIPHER roots when $r = 22$ is $2^{22.04}$.

Algorithm 8 finds the permutation roots of the l-cycles of $(P \circ K)^r$

Require: an ARRAY to hold all the roots of l-cycles of $(P \circ K)^r$
Require: $G(l, a_l)$, $X(l, a_l)$, $k = |G(l, a_l)|$
Ensure: possible candidates for the roots of l-cycles.

for $a_l \neq 0$ in $(P.K)^r$ **do**
  for each $x_j = \{x_{j1}, x_{j2}, \ldots, x_{jk}\} \in X(l, a_l)$ **do**
    for each *cycle* $\in a_l$ cycles of length l **do**
      Select the first i such that $x_{ji} > 0$
      Invoke Alg. 7 to find x_{ji} cycles of length $g_i l$ (current = first element in *cycle*)
      Save the roots found in the last step in the roots ARRAY
      if the number of roots found in step 5 = 0 **then**
        goto step 2
      else
        for i = i + 1 → k **do**
          prevARRAY = ARRAY
          ARRAY = {}
          if $x_{ji} \neq 0$ **then**
            for $e \in$ prevARRAY **do**
              Delete all the cycles involved in e from the l-cycles of $(P.K)^r$
              for each *cycle* $\in$ currently available l-cycles **do**
                Invoke Alg. 7 to find x_{ji} cycles of length $g_i l$ (current = first element in *cycle*)
                Concatenate each root found in the last step with the current element
                for each concatenation **do**
                  if all the 3-bit permutations in the concatenated cycles are valid **then**
                    Add the concatenated cycles as a one element to ARRAY
                  end if
                end for
              end for
            end for
          end if
        end for
      end if
    end for
  end for
end for

Table 5.1: Results of the 10^4 trials and the worst case for $2 \leq r \leq 22$, $n_k \equiv$ the number of keys that yield more than one PRINTCIPHER permutation root, $n_w \equiv$ the number of PRINTCIPHER permutation roots in the worst case.

r	$\log_2 n_k$	$\log_2 n_w$	r	$\log_2 n_k$	$\log_2 n_w$	r	$\log_2 n_k$	$\log_2 n_w$
2	-	-	9	7.66	8.58	16	10.80	18.95
3	-	-	10	8.33	11.90	17	8.71	-
4	6.11	2	11	7.94	9.31	18	11.16	20.67
5	2	-	12	11.46	17.39	19	8.77	-
6	9.30	4.17	13	8.47	-	20	10.68	21.54
7	3.70	-	14	9.10	16.27	21	9.18	18.73
8	9.59	10.07	15	9.77	16.63	22	9.59	22.04

These roots are found within less than 3 hours on a standard PC.

Furthermore, we tried 10^4 random PRINTCIPHER-48 permutation keys excluding the ones that yield $(P \circ K)^r = \textit{Identity}$. Note that, for a random key, the probability for the worst case is $\frac{2^{22.04}}{2^{32}} = 0.001$ for 22 rounds and less than that for $r < 22$. These experiments took a few seconds on average on a standard PC and they show that most of the time there is a unique PRINTCIPHER permutation root. Table 5.1 shows the number of keys (n_k), out of the 10^4 random keys, that yield more than one PRINTCIPHER permutation root. It also shows the number of PRINTCIPHER permutation roots in the worst case (n_w) for each number of rounds.

5.4 Security Based on the Many r-th Roots of a Permutation

Almost all the current authentication protocols base their security proofs on a one-way cryptographic function. In contrast, we propose an authentication protocol whose security is based on a multi-valued function which is interestingly the roots of a permutation π in the group of all permutations S_n.

From the previous sections, we know that the number of r-th roots of a permutation $(P \circ K)^r$ can be very large. We also know that we are only able to recover the PRINTCIPHER permutation key K because we know the structure of K and the whole permutation P. However, if the structure of K is unknown, then the above attack would not work. Based on these two observations we can construct the following authentication protocol.

An Authentication Protocol Based on a Multi-Valued Function: Assume that there are two parties P (Prover) and V (Verifier) who share a secret key $K \in S_n$. P wants to prove to V that it knows the secret key K. Our authentication protocol can be described as follows:

1. V generates a random permutation $C_p \in S_n$.
2. V sends a challenge $C_p \in S_n$ to P.
3. P generates a random permutation $C_v \in S_n$ and computes $Q = (C_v \circ K \circ C_p)^r$, where r is a composite number chosen in a specific way (If r is prime, the number of roots could possibly be smaller than when it is composite).
4. P sends Q, C_v and r to V (we can hide r to add more security at the cost of increasing the verification time for V).
5. V checks whether $(C_v \circ K \circ C_p)^r$ equals Q or not (if r is not sent V starts by trying $r = 2$ and incrementing r until a match is found).
6. If it equals then "P knows K, Success" else "P doesn't know K, failure".

Security Analysis: The obvious way for an adversary to recover K, is to compute the r-th root of Q. Computing the r-th root of Q is easy but there could be many roots. If $Q \in S_{64}$ then depending on the choice of r, there can be more than 2^{64} roots and sometimes more than 2^{128}. Thus if our security parameter is 64-bit or 80-bit, the protocol can be secure against this specific attack. Therefore, we need to choose r so that $(C_v \circ K \circ C_p)^r$ has r-th roots more than 2^s where s is our security parameter. Using formula 5.2 for computing the number of roots, we fix the number of 1-cycles (a_1) and r and check the corresponding number of roots. We varied both a_1 from 20 to 32 and r from 4 to 20, and calculated the corresponding number of roots resulting only from the 1-cycles. The following table shows the minimum values of r and a_1 that gives us at least 2^{64} or 2^{80} number of rounds.

The above table shows that when $r \geq 4$ and the number of 1-cycles of the permutation $Q = (C_v \circ K \circ C_p)^r$ is ≥ 30 then the number of the rth roots of Q is $\geq 2^{80}$. Thus, it would be infeasible to compute the r-th roots of Q in order to find the secret key K. This might be one step towards proving that the protocol is secure. However, as pointed by Shamir in any challenge-response application the secret key (here K) and the challenge (here C_p) should be securely mixed similar to the mixing of the secret and IV in stream ciphers and then base the response on the result of the mixing function [105].

Table 5.2: The table shows the minimum combination of both r and a_1 that makes the number of rth roots of a permutation consisting only of cycles with length 1 at least 2^{64} or 2^{80}. Note that r is a composite number as for some prime numbers, the number of r-th roots is less than 2^{80} or 2^{64}.

min(r)	min(a_1)	min(# of r-th roots)
4	30	2^{80}
4	26	2^{64}
6	27	2^{80}
6	23	2^{64}
15	26	2^{80}

Therefore to avoid passive attacks that might exploit the algebraic structure of the permutation $C_v \circ K \circ C_p$ or active attacks that might fix C_p to a certain permutation such as the *Identity* permutation and exploit the algebraic structure of $C_v \circ K$ — for example by fixing the structure of K to a certain class of permutations such as that of PRINTCIPHER, we need to use a simple mixing function $\text{Mix}(C_v, K, C_p)$ that yields a permutation M and then return $Q = M^r$ as a response to the challenge C_p. The additional random value C_v is generated mainly to prevent replay attacks.

More research needs to be done in order to specify a simple and suitable mixing function that yields a permutation. Note that a permutation in S_n where n is a power of 2 can be represented as an $(\log_2 n)$-bit Sbox in hardware [73]. Note also that the inputs to the mixing function do not need to be permutations but only the output needs to be a permutation in S_n. We believe that finding a suitable mixing function beside a hardware implementation for this protocol are two interesting topics for future work.

5.5 Conclusions

We have described two differential attacks against 22 rounds of PRINTCIPHER-48, requiring the full code book and about 2^{48} computational steps. Similar results can be obtained for the 96-bit version of the cipher.

One of the attacks is a new technique targeting the key-dependent permutations used in PRINTCIPHER. Since such key-dependent permutations are currently not well-studied, the attack is of importance to past and future designs that use them. We introduced a novel technique for computing permutation roots, making it possible to retrieve the key-dependent single-round permutation π given nothing but the r-round permutation π^r and the cipher description. While our technique so far applies only to the case where the linear layer is a (key-

depended) bit permutation, future designers of cryptographic primitives using key-dependent permutations should be aware of this technique when choosing parameters like round numbers or Sbox layout for their algorithms.

We also showed how to design an authentication protocol whose security is based on a multi-valued function, namely, the r-th roots of a permutation in the symmetric group S_n.

Chapter 6

Cryptanalysis of PRINTcipher: The Invariant Subspace Attack

In this chapter, we present a new attack called *invariant subspace attack* that breaks the full cipher for a significant fraction of its keys. For such weak keys, a chosen plaintext distinguishing attack can be mounted in unit time. This new attack can be seen as a weak-key variant of a statistical saturation attack. We show that for weak keys, strongly biased linear approximations exists for any number of rounds. In this sense, PRINTcipher behaves very differently to what is usually – often implicitly – assumed.

In a nutshell, the attack is based on the observation that for PRINTcipher there exist cosets of subspaces of $\mathbb{F}_2^n$ that the round function maps to cosets of the same subspace. The exact coset is determined by the round key only. Now, if the round key is such that a coset gets mapped to itself, the fact that all round keys are identical in PRINTcipher (almost) immediately leads to the conclusion that for certain (weak) keys some affine subspaces are invariant under encryption. The round constants, mainly introduced to avoid slide attacks, do not prevent the attack as the round constants are included in the subspace. The principle of the attack is described in Section 6.1.1.

More particular, using this new attack technique, which we call (for obvious reasons) *invariant subspace attack*, we demonstrate the existence of 2^{52} weak keys (out of 2^{80}) for PRINTcipher-48 and 2^{102} weak keys (out of 2^{160}) for PRINTcipher-96. If a key is weak, our attack results in a distinguisher using less than 5 chosen plain- or ciphertexts. That is, even in the case of RFID-tags, where the amount of data available for a practical attack is strictly limited, our attacks apply. In a known plain- or ciphertext scenario the data complexity increases by a factor of 2^{16} (PRINTcipher-48) resp. 2^{32} (PRINTcipher-96).

Besides the low data complexity of the distinguisher, the attack technique has interesting relations to statistical saturation attacks and truncated differential attacks. In this chapter, we are concerned only about its relation to the statistical saturation attack. For more information on its relation to the truncated differential attack, we refer the reader to the original paper [83].

It follows in particular that PRINTcipher is an example of a non-toy cipher where attacks

do not behave as we usually expect them to. The bias for statistical saturation attacks and the bias of linear hulls are extremely key-dependent. For a weak key, increasing the number of rounds up to the full number of rounds does not increase the security of the cipher with respect to these attacks.

6.1 The Invariant Subspace Attack

6.1.1 General Idea

Consider an n-bit block cipher with a round function E_k consisting of a key addition and an SP-layer

$$E : \mathbb{F}_2^n \to \mathbb{F}_2^n,$$

that is E_k is defined by $E_k(x) = E(x + k)$. Assume that the SP-layer E is such that there exists a subspace $U \subseteq \mathbb{F}_2^n$ and two constants $c, d \in \mathbb{F}_2^n$ with the property:

$$E(U + c) = U + d.$$

Then, given a (round) key $k = u + c + d$ with $u \in U$, the following holds:

$$E_k(U + d) = E((U + d) + (u + c + d)) = E(U + c) = U + d,$$

i.e. the round function maps the affine subspace $U + d$ onto itself. If all round keys are in $k \in U + (c + d)$ (in particular if a constant round key is used), then this property is iterative over an arbitrary number of rounds. This yields a very efficient distinguisher for a fraction of the keys. U should be as large as possible to increase this fraction. We call this new attack technique an *invariant subspace attack*. In the next section we show an example of how to apply it to the lightweight block cipher PRINTcipher.

6.1.2 Attack against PRINTcipher

An Attack on PRINTcipher

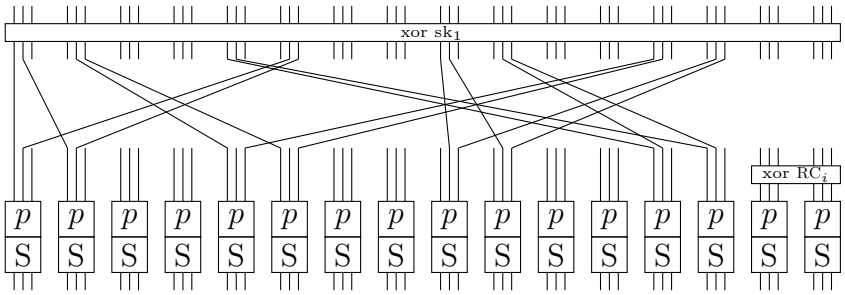
One interesting property of the PRINTcipher Sbox is that a one bit difference in the input causes a one bit difference in the same bit in the output with probability $\frac{2}{8}$. That is, there exists exactly one pair for each one bit input difference resulting in a one bit output difference (at the same position). More precisely, denoting by $*$ an arbitrary value in $\mathbb{F}_2$, the following holds for the PRINTcipher Sbox:

Table 6.1: Specific 1-bit input and 1-bit output differences with probability 1

S(000) = 000 S(001) = 001	$\Leftrightarrow$	S(00*) = 00*
S(100) = 111 S(110) = 101	$\Leftrightarrow$	S(1*0) = 1*1
S(011) = 110 S(111) = 010	$\Leftrightarrow$	S(*11) = *10

In addition, there exists a subset of Sboxes such that (1) two output bits of those Sboxes map onto two input bits of the same Sboxes in the next round and (2) the round-dependent RC_i is not involved (See Figure 6.1).

Fig. 6.1: A subset of PRINTcipher-48 Sboxes mapping onto itself.



Now consider an xor-key sk_1 of the form

Xor key = `01* *11 *** *** 01* *11 *** *** 01* *11 *** *** 01* *11 *** ***`,

and a permutation key with the following restrictions:

Perm. key = `0* 11 ** ** 10 01 ** ** 11 *0 ** ** *0 11 ** **`,

where again $*$ denotes an arbitrary value in $\mathbb{F}_2$. For those keys the following structural *iterative* one round property holds:

Start	`00* *10 *** *** 00* *10 *** *** 00* *10 *** *** 00* *10 *** ***`
Key xoring	`01* *01 *** *** 01* *01 *** *** 01* *01 *** *** 01* *01 *** ***`
Lin. layer	`00* 11* *** *** 0*0 1*1 *** *** *00 *11 *** *** 00* 11* *** ***`
RC	`00* 11* *** *** 0*0 1*1 *** *** *00 *11 *** *** 00* 11* *** ***`
Perm. layer	`00* *11 *** *** 00* *11 *** *** 00* *11 *** *** 00* *11 *** ***`
Sbox layer	`00* *10 *** *** 00* *10 *** *** 00* *10 *** *** 00* *10 *** ***`

This property holds with probability one if both keys are of the above form. The fraction of those keys is 2^{-16} for the XOR key and 2^{-13} for the permutation key, meaning that the property is met for a fraction of (2^{-29} of all keys. In other words, there exist 2^{51} weak keys of this form.

Thus, one can very efficiently check if a key of the above form is used by encrypting a few texts of the above form and check if the ciphertext is again of the same form. Given that the probability for false positives is $\approx 2^{-16}$, trial encrypting just a handful of selected plaintexts will uniquely identify such a weak key. If such a key is found, we do of course immediately have a distinguisher on PRINTcipher.

Invariant Subspace Description:

Let us briefly rephrase the attack in terms of an invariant subspace attack. For this we fix a permutation key of the above form. Remember that the inner state at the beginning and the end of each round was

$$\text{Start} = \texttt{00* *10 *** *** 00* *10 *** *** 00* *10 *** *** 00* *10 *** ***}.$$

This means that the relevant subspace $U \subset \mathbb{F}_2^{48}$ is defined by

$$U = \{\texttt{00* *00 *** *** 00* *00 *** *** 00* *00 *** *** 00* *00 *** ***}\}, \quad (6.1)$$

and that the affine subspace is defined by any fixed vector d of the form

$$d = \texttt{00* *10 *** *** 00* *10 *** *** 00* *10 *** *** 00* *10 *** ***}. \quad (6.2)$$

Then for any fixed vector c of the form

$$c = \texttt{01* *01 *** *** 01* *01 *** *** 01* *01 *** *** 01* *01 *** ***}, \quad (6.3)$$

and any xor-key $k \in (U + c + d)$, the round function does indeed map $U + d$ onto itself.

6.1.3 Other Attack Profiles

In the following we describe other sets of weak-keys for PRINTcipher-48 and similar ones for PRINTcipher-96.

Other weak keys for PRINTcipher-48

As it turns out, there are some more invariant subspaces that also can be used for PRINTcipher-48. They are all of the form

```
00* XXX *** 1*1 00* *10 *** *** 00* XXX *** 1*1 00* *10 *** ***,
```

where an 'X' marks a bit position where the attacker has to make an arbitrary assignment. Note that each position can be filled independently of the others. Thus, we have 2^6 possible plaintexts that we can work with, each of which targets another class of weak keys.
For each such assignment, the cipher behaves as follows:

Start	(1)	`00* XXX *** 1*1 00* *10 *** *** 00* XXX *** 1*1 00* *10 *** ***`
Key xoring	(2)	`0X* X01 *** X*1 01* *0X *** *** 0X* 001 *** X*X 01* *0X *** ***`
Lin. layer	(3)	`00* XXX *** X*X 0*0 1*1 *** *** *00 XXX *** 10* 00* 11* *** ***`
RC	(4)	`00* XXX *** X*X 0*0 1*1 *** *** *00 XXX *** 10* 00* 11* *** ***`
Perm. layer	(5)	`00* XXX *** 1*0 00* *11 *** *** 00* XXX *** 1*0 00* *11 *** ***`
Sbox layer	(6)	`00* XXX *** 1*1 00* *10 *** *** 00* XXX *** 1*1 00* *10 *** ***`

The behaviour is best understood by traversing the cipher in the inverse direction, i.e. by starting from the end and then finding the key bits that ensure that all fixed bits in line (1) match their counterparts in line (6).

Let us start with the output of the Sbox, i.e. line (6), and let the bit positions marked by 'X' be arbitrarily and independently be fixed to either 0 or 1. Then going backwards through the Sbox uniquely determines the bits in line (5). We then use a permutation key of the form

Perm. Key = `0* ** ** (00 or 11) 10 01 ** ** 11 ** ** 10 0* 11 ** **`

to obtain line (4), noting that 2^{-13} of all permutation keys meet this property. We then apply round counter and linear layer to obtain line (2). Now note that line (2) contains 22 bits that are fixed and that have to match the corresponding bits in line (1). Thus, 22 key bits of the xoring key are determined, meaning that 2^{-22} of all xoring keys are suitable for the attack.

Summing up, for each of the 2^6 possible assignments to the bits marked by 'X' in line (1) or (6), a fraction of exactly 2^{-35} keys are weak, meaning that in total, we have found another fraction of 2^{-29} weak keys that can be attacked by the above technique.

Analysis of PRINTcipher-96

As it turns out, the same attack can also be applied to PRINTcipher-96. Again, there are two types of weak keys. The first type is based on 32 active bits and is met by a fraction of

2^{-59} of all keys. The second type is based on 44 active bits and has an additional 12 freely chosable input bits. Each of the resulting 2^{12} inputs targets a fraction of 2^{-71} keys, meaning that this group, too, contains a fraction of 2^{-59} weak keys in total. The active bits for these weak keys are given in Table 6.2.

Subset 1	**Active input bits for linear layer:** (0 1) (4 5) (12 13) (16 17) (24 25) (28 29) (36 37) (40 41) (48 49) (52 53) (60 61) (64 65) (72 73) (76 77) (84 85) (88 89)
	Active output bits for linear layer: (0 2) (3 5) (12 13) (15 16) (25 26) (28 29) (36 38) (39 41) (48 49) (51 52) (61 62) (64 65) (72 74) (75 77) (84 85) (87 88)
Subset 2	**Active input bits for linear layer:** (0 1) (3 4 5) (9 11) (12 13) (16 17) (24 25) (27 28 29) (33 35) (36 37) (40 41) (48 49) (51 52 53) (57 59) (60 61) (64 65) (72 73) (75 76 77) (81 83) (84 85) (88 89)
	Active output bits for linear layer: (0 2) (3 4 5) (9 10) (12 13) (15 16) (25 26) (27 28 29) (33 35) (36 38) (39 41) (48 49) (51 52 53) (58 59) (61 62) (64 65) (72 74) (75 76 77) (81 82) (84 85) (87 88)

Table 6.2: Subsets of active bits for PRINTcipher-96, grouped according to Sboxes

6.1.4 Protecting Against the Attack

The above attack against PRINTcipher is a special case of the general attack described in the beginning of the section, since the subspace is described by simply fixing some of its bits. In theory, describing the subspace by a set of linear equations is possible, opening for a wide range of attacks. The full potential of this generalized attack is yet to be determined.

As for the special case used against PRINTcipher, it is relatively easy to protect the design against the attack. Note that the list of attack profiles by fixing bits given here is complete, and that all attack profiles fix two of the bits 39-41 (PRINTcipher-48) resp. 87-89 (PRINTcipher-96). Thus, it would suffice to spread the round counter over the last three Sboxes, e.g. by assigning two counter bits to each Sbox. This would destroy the only attack profiles available.

We also analyzed the block cipher NOEKEON, which was proposed by Daemen et al. in 2000 [42]. NOEKEON is a 16-round block cipher with a constant round key, making it a particularly tempting target for the attack. However, as it turns out, the linear mixing layer of NOEKEON is much more resistant against the above type of attack. Here, the stronger

round function (necessary for a cipher with only 16 rounds) works to the advantage of the cipher. As it turns out, even if there was no round counter involved in NOEKEON, the simple attack described above – i.e. where the subspace is defined by fixing certain bits – could not be applied. Whether or not the generalized attack has a better chance of succeeding remains yet to be determined.

6.2 Statistical Saturation Attacks

The attack on PRINTCIPHER discussed in Section 6.1.2 is clearly strongly related to statistical saturation attacks described in Chapter 2. In this section, we show that for PRINTCIPHER there exist strongly biased linear approximations for any number of rounds, if the key is weak in the sense of the invariant subspace attack. This result follows using a link between statistical saturation attacks and multi-dimensional linear attacks (see Chapter 2 or [82]).

6.2.1 On the Choice of the Values of the Fixed Bits

We use the statistical saturation attack as described in Chapter 2 where the encryption function $e : \mathbb{F}_2^r \times \mathbb{F}_2^s \to \mathbb{F}_2^t \times \mathbb{F}_2^u$ defined in equation (2.8) is restricted to $h_y : \mathbb{F}_2^s \to \mathbb{F}_2^t$ defined in equation (2.9). We now focus on the case where $r = t$, that is the number of fixed bits is the same as the number of bits considered at the output.

Assume that we have a cipher 'e' that is vulnerable to an invariant subspace attack. As for statistical saturation attacks, up to a fixed bijective linear transformation before and after the cipher, we can assume that, for a weak-key, the affine subspace of the form $\{d\} \times \mathbb{F}_2^s$ is mapped to a affine subspace of the form $\{d\} \times \mathbb{F}_2^s$. It then follows immediately that (for a weak key) the function of the restriction h_y for $y = d$ is a constant, more precisely

$$h_d(x) = e^{(1)}(d, x) = d \tag{6.4}$$

For the special choice of the values of the fixed bits the capacity of the probability of the output distribution is maximal. Hence for a weak key this special fixing of the bits leads to an optimal statistical saturation attack. Note that Theorem 2.2.4 does not reveal the existence of such extreme cases, as it only considers the average capacity of the restrictions.

While in an invariant subspace attack, given the subspace, the choice of the coset is crucial, for statistical saturation attacks the fixed bits are usually assigned with random values. As the invariant subspace attack on PRINTCIPHER does not imply that PRINTCIPHER is in general

vulnerable to a statistical saturation attack, it does not come as a surprise that the experiments in [78] did not reveal any weakness of PRINTCIPHER with respect to those attacks.

6.2.2 On the Existence of Highly Biased Approximations

The following corollary follows from Identity (2.10).

Corollary 6.2.1. *Assume an n-bit block cipher E_k is vulnerable to an invariant subspace attack, that is there exist a subspace U, a constant d and keys k such that*

$$E_k(U + d) = U + d.$$

Then, for those keys, there exist linear approximations with a correlation c such that

$$c \geq 2^{\dim(U)-n} - 2^{2(\dim(U)-n)}.$$

Proof. Let p_y be the the probability distribution of the r output bits of h_y and let also p_d be the the probability distribution of the r output bits of h_d. As defined by equation (6.4), h_d is a constant function. Thus $\mathrm{Cap}(p_d) = 2^r - 1$ and furthermore

$$\sum_{y \in \mathbb{F}_2^r} \mathrm{Cap}(p_y) \geq \mathrm{Cap}(p_d) = 2^r - 1.$$

Considering Identity (2.10) it follows that

$$\sum_{\substack{a \in \mathbb{F}_2^r \times \{0\} \\ b \in \mathbb{F}_2^t \times \{0\}, b \neq 0}} (\widehat{e}(a,b))^2 \geq 2^{2n}(1 - 2^{-r})$$

Lower bounding the maximal value by the average (and recalling that $r = t$), we compute

$$\max_{a,b \neq 0} (\widehat{e}(a,b))^2 \quad \geq \quad 2^{-2r} \sum_{\substack{a \in \mathbb{F}_2^r \times \{0\} \\ b \in \mathbb{F}_2^t \times \{0\}, b \neq 0}} (\widehat{e}(a,b))^2 \geq 2^{2n-2r}(1 - 2^{-r})$$

Thus there exists at least one Fourier coefficient such that

$$|\widehat{e}(a,b)| \geq 2^{n-r}\sqrt{1 - 2^{-r}} \geq 2^{n-r} - 2^{n-2r}$$

Applying identity (2.1) and remembering that $r = n - \dim U$, the theorem follows.

□

Clearly, this Corollary is only interesting for the case where $\dim(U) > \frac{n}{2}$ as the existence of the stated approximations otherwise is trivial. For the case of PRINTCIPHER-48 the following Corollary holds.

Corollary 6.2.2. *Given a weak key for any round $r \leq 48$ there exists at least one linear approximation for* PRINTCIPHER*-48 with correlation at least $2^{-16} - 2^{-32}$.*

Understanding the Large Correlations

Here we summarize the analysis described in [2] about the above large correlations which are caused by the invariant subspace. One can see that the linear approximations with large correlations in PRINTCIPHER have input masks α and output masks β that lie in the orthogonal subspace of U, i.e. $\alpha, \beta \in U^{\perp}$. This is because the orthogonal of the subspace U defined in 6.1,

$$U^{\perp} = \{\texttt{**0 0** 000 000 **0 0** 000 000 **0 0** 000 000 **0 0** 000 000}\}$$

corresponds to the plaintext and ciphertext bits that are fixed in the invariant subspace.

Thus to study the large correlations of these linear approximations, the work in [2] split their linear hull represented by Equation (2.2) into two sums where the first sum corresponds to the linear trails with intermediate masks residing in $U^{\perp}$ and the second sum corresponds to the other linear trails with intermediate masks outside $U^{\perp}$ for at least one round. Thus for an iterated block cipher F,

$$C_F(\alpha, \beta) = \sum_{\alpha_i \in U^{\perp}} (-1)^{s_i \oplus d_i} |C_i| + \sum_{\alpha_i \notin U^{\perp}} (-1)^{s_i \oplus d_i} |C_i| \tag{6.5}$$

Now, let M_i be the correlation matrix of the ith round function of an iterated block cipher with identical round keys and small round constants. Let A be the submatrix of M_i whose entries correspond to input and output masks in $U^{\perp}$ and are not affected by the round constants. Note that since the round constants do not affect the entries of A, so we have the same submatrix for each round in PRINTCIPHER. Thus, A^r describes exactly the contribution to the linear hull from the linear trails with intermediate masks residing in $U^{\perp}$. If the entries of A^r have a large magnitude, then the corresponding elements of the product $M_r M_{r-1} \cdots M_1$ have more or less the same magnitude, unless the contributions from the second sum cancel those from the first sum.

As the asymptotic behaviour of A^r is related to the spectrum of A [29, 94], the work in [2] studied the eigenvalues of A. Define $v = (v_\alpha)_{\alpha \in U^{\perp}}$ where $v_\alpha = (-1)^{\langle d, \alpha \rangle}$. It was shown in [2]

that v^T is an eigenvector with eigenvalue 1 iff $F(U+d) = U+d$ where F is an invertible vectorial Boolean function. Moreover, it was shown that in the case where there is no other non-trivial[1] eigenvector with eigenvalue 1, the sequence A^r converges to $(2^{\dim(U^\perp)}-1)^{-1}v^T v$ as $r \to \infty$. Thus in equation 6.5, if the contribution to $C_F(\alpha,\beta)$ from the second sum are negligible, then all trails with nonzero $\alpha, \beta \in U^\perp$ have absolute correlations $\approx 2^{-\dim(U^\perp)}$.

Furthermore, experiments were performed in [2] to verify that PRINTCIPHER behaves according to the above stated results. The submatrix A of PRINTCIPHER-48 was constructed according to the class of weak keys described in Section 6.1.2 and it was verified that $Av^T = v^T$. Also 16 linear approximations with absolute correlations approximately equal to 2^{-16} were found. For more information about these results, we refer the reader to [2, 5].

6.3 Conclusions

We have presented a new attack against iterative block ciphers named *invariant subspace attack* and demonstrated its validity by breaking PRINTCIPHER for a significant fraction of its keys. The presented *invariant subspace attack* verifies that there exist 2^{52} weak keys of the 2^{80} possible keys for PRINTCIPHER-48 and 2^{102} weak keys of the 2^{160} possible keys of PRINTCIPHER-96. Note that as pointed in Chapter 3 that all the classes of weak keys that lead to invariant subspaces have been identified in [30].

Moreover, we have shown that the *invariant subspace attack* has relationships to other classes of attacks as in multi-dimensional attack linear attack and statistical saturation attack. This link shows that for PRINTCIPHER there are strongly biased linear approximations with absolute correlation at least 2^{-16} for any number of rounds, if a weak key is chosen. We have also given a brief description about the work in [2] which mainly explains why the invariant subspace causes these strongly biased linear approximations.

[1] Note that $(1,0,0,\ldots,0)^T$ is an eigenvector of A with eigenvalue 1.

Chapter 7

Cryptanalysis of the Lightweight Cipher A2U2

In this chapter, we present a cryptanalysis of the stream cipher A2U2 described in Chapter 3. We start by describing some useful properties used to attack the cipher (Section 7.1) and by showing how to repair the nonfunctional chosen plaintext attack on A2U2 described in Chapter 3 (See Section 3.3.3), yielding an attack for the chosen plaintext case that requires less than a second running time and very little data (Section 7.2).

We then proceed to discuss the more challenging class of known-plaintext attacks. In Section 7.3 we describe a guess-and-determine attack that requires guessing 2^{49} inner state bits in order to retrieve the starting state.

In addition, we present a chosen-IV attack that makes use of the special key/IV setup of the cipher. A special design feature of A2U2 is that the number of initialization rounds varies and depends on an internal counter. The number of rounds varies from 9 to 126. In Section 7.4, we propose a differential-style attack which first enables us to find the 5-bit counter key. Moreover, we present an attack that recovers the master key in the case that only 9 initialization rounds are used. The latter attack requires only 2^{38} computational steps.

Finally, in Section 7.5, we describe urgent changes that should be done to the cipher design in order to avoid the above attacks. We also give some further research directions, both for cryptanalysis and improvement of the cipher.

In this chapter, we use the notations and algebraic equations described previously in Chapter 3 (See Section 3.1.3).

7.1 Useful Properties Used in the Attacks

In the following, we point out some properties of the cipher that will be used by our attacks.

Known Counter: Since the counter has the all-one state after initialization, the attacker can compute all successive counter states and does know the bit C_t for any $t \geq 0$. This simplifies significantly some of the algebraic equations described in Chapter 3 (See Section 3.1.3).

Chosen Randomization Vector: Instead of using a nonce that is chosen solely by the encrypting party as is common practice with stream ciphers, the random vector used for initialization is generated in collaboration between sender and receiver. According to the design specification of the cipher [46], both sender and receiver each choose a 32-bit random number which is then sent over the communication channel. Both inputs are then combined by xor into the actual randomization vector. Note that this procedure enables an active attacker to choose the randomization vector: He waits for the legitimate party's input and chooses his own such that the xor sum will be the desired number. In particular, he can introduce arbitrary differences between randomization vectors and even force the encryption device to use the same randomization vector twice, thus violating an important design principle for stream ciphers.

Chosen Plaintext: Due to the unusual output generation, a chosen plaintext attack is more powerful than a known plaintext attack. This is another difference to traditional stream cipher designs, where no extra information is gained if the attacker is allowed to choose the plaintext himself.

7.2 A Chosen Plaintext Attack

7.2.1 A Leak in the Output Function

In the following, we will demonstrate a leak in the output function that can even be used for general known-plaintext attacks and will then expand this weakness into a chosen plaintext attack that reminds of the one described in Chapter 3 (See Section 3.3.3) but is actually functional.

Known plaintext: Assume that the inner sequences A_t and B_t are statistically close to random. Then in particular, $\Pr(A_t = 0) = \frac{1}{2}$ for all t. We can now consider two cases for the output function:

- If $A_t = 0$, then $Y_t = B_t + C_t$. Since we know C_t, we can rewrite this as $B_t = Y_t + C_t$. For $A_t = 0$, this is always true.
- If $A_t = 1$, we have $Y_t = B_t + P_{\sigma(t)}$, with $P_{\sigma(t)}$ unknown. If we assume that $P_{\sigma(t)} = C_t$ with probability $\frac{1}{2}$,[1] then the equation $B_t = Y_t + C_t$ is also true with probability $\frac{1}{2}$.

In total, the equation $B_t = Y_t + C_t$ is thus met with probability $\frac{1}{2} + \left(\frac{1}{2}\right)^2 = \frac{3}{4}$, i.e. by observing the keystream and knowing the behaviour of the counter LFSR, we can predict the inner stream $(B_0, B_1 \ldots)$ with probability $\frac{3}{4}$ per bit.

Chosen plaintext: Note that when we can choose the plaintext, we can increase the probability of $P_{\sigma(t)} = C_t$ and thus the probability of the equation $B_t = Y_t + C_t$ being correct.

As an example, consider the first 5 output bits of the LFSR, which are $(1, 0, 0, 0, 0)$. Thus, if we choose a plaintext $(1, 0, 0, 0, 0)$, then $P_{\sigma(t)} = C_t$ is true with probability 1 for the first bit, $1 - \frac{1}{2}$ for the second, $1 - \frac{1}{4}$ for the third, $1 - \frac{1}{8}$ for the fourth, and $1 - \frac{1}{16}$ for the fifth bit. Thus, we can predict the inner state bits $(B_0, \ldots, B_4)$ with probabilities $1, \frac{3}{4}, \frac{7}{8}, \frac{15}{16}, \frac{31}{32}$.

7.2.2 The Attack

The most useful plaintexts for this kind of analysis seem to be $(0, 0, ...)$ and $(1, 1, ...)$, since for them, the attacker knows exactly the bit $P_{\sigma(t)}$ for every time slot t. Let us start with the all-zero sequence. The attacker knows that the plaintext sequence $(P_{\sigma(t)})_{t \geq 0}$ consists only of zeros. He now looks at all time slots t with $C_t = 0$. For those time slots, it holds that $B_t = Y_t$, independent of the choice of A_t. Thus, he learns about half of the bits of the sequence B.

The remaining bits can be learned using the all-one sequence. In this case, in all positions where $C_t = 1$, the attacker learns $B_t = Y_t + 1$, also independent of A_t. Thus, he has fully reconstructed the sequence B.

What is more, he can also use this new information to learn the sequence A as well. For every time slot, he picks the ciphertext bit Y_t corresponding to the plaintext bit $P_{\sigma(t)} \neq C_t$. If it holds that $B_t = Y_t + C_t$, then $A_t = 0$, otherwise $A_t = 1$.

After this step, the attacker knows the sequences generated by all three registers A, B, and C. The remaining attack proceeds as follows. Knowing the sequences A, B and C the attacker can determine the values h_t because

$$h_t = B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + A_t + 1.$$

[1] Note that if the probability for $P_{\sigma(t)} = C_t$ significantly differs from $\frac{1}{2}$, then the success probabilities for the rest of the attack are even better than claimed.

Furthermore, it holds that

$$h_t = \mathrm{MUX}_{C_{t-5}}(S_0^t, S_1^t) \cdot \mathrm{MUX}_{C_{t-1}}(S_4^t, A_{t-2}) + \mathrm{MUX}_{C_{t-3}}(S_2^t, S_3^t) + 1,$$

where C_{t-i} for $i = 1, 3, 5$ and A_{t-2} are known. This equation is at most quadratic and in about half of the cases (when $C_{t-1} = 1$) it is linear. Determining 56 values of h_t yields a fully determined quadratic Boolean equation system, which can be solved by e.g., using Gröbner basis techniques. As about half of the equation are linear, a linear equation system can be obtained after determining 112 values of h_t. After 11 clockings of the algorithm the key register is rotated once and the key bits are reused, thus it can happen that the same equation is generated twice. However, experiments showed that this does not happen frequently, thus we expect that observing around 120 values of h_t is sufficient to generate a fully determined linear equation systems in 56 unknowns.

Effort: The attack requires two chosen plaintexts of length around 60 bits which are encrypted using the same key and initialization vector in order to recover secret key bits (excluding the 5 bits which are used to initialize the counter). Note that a 60 bit plaintext corresponds to a ciphertext of length 120 bit on average. The main computational effort consists in solving a linear Boolean equation system which can be done in well under a 1 second. Thus, in the chosen plaintext scenario, the cipher must be considered as completely broken.

7.3 Guess-and-Determine Attack

In this section we discuss a known plaintext attack which is in general a more likely scenario than a chosen plaintext attack. When we know but are not allowed to choose the plaintext we cannot use the same trick as in Section 7.2 to determine the sequence $(B_t)_{t\geq 0}$. We cannot simply calculate this sequence for a given plaintext/ciphertext pair because we do not know which bits of the ciphertext correspond to the plaintext bits. This is controlled by register A. The idea of this attack is to guess the sequence of $(A_t)_{t\geq 0}$, meaning we guess at which positions of the ciphertext a plaintext bit was used. These guesses are used to determine additional bits of register B and then later on the value of h_t. As we know the value of the counter at any time during the encryption process, given h_t and A_{t-2} we obtain a Boolean equation in the key bits which is at most quadratic and contains at most three variables. If we are able to collect sufficiently many of those equations we will be able to recover the key bits by solving the equation system.

We denote by A_0 the content of the last cell of the first NFSR at the time when the ciphertext

generation starts. The attack is divided into three parts of guessing bits.

In the first part we guess the value A_t for $t = 0, \ldots, 8$ for 9 consecutive clockings of the algorithms. Depending on our guess we know if the counter bit or a plaintext bit was used to generate the corresponding ciphertext bit and we can determine the value of B_t for $t = 0, \ldots, 8$. After guessing 9 bits we know the full second NFSR and about the lower half of the first NFSR.

In the next part we continue guessing the value of A_t for $t = 9, \ldots, 16$ and determine the value of B_t for $t = 9, \ldots, 16$. Additionally, we obtain the value of h_t for $t = 9, \ldots, 16$ and the corresponding Boolean equation in the key bits, because it holds that

$$h_t := A_t + B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + 1, \tag{7.1}$$

the full register B is known and we guessed the value of A_t. After the second part of the attack both registers are known and we have already obtained 8 equations.

In the third part we want to determine the value of h_t for further clockings of the cipher. The full register A is known and in order to update register B only bits of register A are used. This means we can update register B and know the value which was use to encrypt next ciphertext bit (bit 17, 18 etc). Furthermore, we know the counter value C_t and the plaintext bit p that might have been used (according to our guess). As mentioned before we want to determine the value of h_t and obtain the corresponding equation. Using equation (7.1) we need to determine A_t in order to obtain h_t. Depending on the values of counter C_t and the value of the plaintext bit $P_{\sigma(t)}$ we can either calculate the value of A_t or we have to guess it. The output generation can be presented as a quadratic equation

$$Y_t + B_t + C_t = A_t(C_t + P_{\sigma(t)}). \tag{7.2}$$

This means if $C_t \neq P_{\sigma(t)}$ and thus $C_t + P_{\sigma(t)} = 1$ we can simply use Equation (7.2) to determine the value of A_t. However, if $C_t = P_{\sigma(t)}$ and thus $C_t + P_{\sigma(t)} = 0$, Equation (7.2) does not yield any information about A_t and we have to guess the value of A_t as before.

Effort: We need to collect at least 56 equations to determine a unique solution in 56 key variables. In the first two parts of the attack we guess 17 bits and obtain 8 equations. We expect that we have to guess every second value for A_t in the third part of the attack. Thus, we expect that we have to guess at least 24 further bits in the third part of the attack in order to obtain a fully determined equation system. This leads to a complexity of at least 2^{41}.

There are two factors that increase the attack complexity. Firstly, the equation system is non-linear and therefore it might not have a unique solution even though it is fully determined.

However, it is often sufficient to add a few extra equations to get a unique solution. This will slightly raise the complexity of the attack.

Secondly, the key register is rotated once after 11 clockings of the algorithm and thus key bits are reused. This property will on the one hand increase the complexity of the attack for the correct guess but on the other hand enable us to discard wrong guesses in an early stage. After producing 11 ciphertext bits the key register has been rotated once, that means the key bits will be reused when generating more equations. This leads to rounds where we guess or determine the value of A_t but do not get a new equation, thus do not gain extra information about the key. This is especially true for the correct guess and means that it is necessary to guess extra bits in order to obtain a fully determined system. For a wrong guess however this might be to our advantage because it is very likely that when the same polynomial is generated the RHS differs. Thus, we get contradicting equations and can abort the guess.

In general, for a wrong guess the equation system will not have a solution. The inconsistency might be very obvious as mentioned above, but it might also be necessary to solve a non-linear Boolean equation system. Therefore, we have to make a trade-off how often we want to check if the system is still solvable.

An implementation of the attack is necessary in order to provide a better estimate of the attack complexity. Simulations showed that after guessing 47 bits we obtain a set with 57 equations on average. When testing these equation systems for solvability around 5% have a non-empty solution set. This means 5% of our guesses survive. Guessing 6 additional bits yields equation systems containing 70.7 equations on average and we expect that only the correct guess or very few guesses survive and that the equation system corresponding to the right guess will have a unique solution. The estimated complexity for this approach is 2^{49} bit guesses. As we in the worst case have to solve an non-linear equation system for each guess we cannot ignore the costs for this step. The costs for solving a non-linear equation system by for example using Gröbner bases are hard to estimate as the problem of solving a random non-linear equation system is NP-hard and thus the running time is equivalent to a brute force search in the worst case. However, we deal with fairly small equation systems and experiments indicate that these equation systems are solvable in a fraction of seconds using Gröbner basis algorithms and that the costs are comparable to about four encryptions. Furthermore, the number of Gröbner basis applications can be reduced by implementing techniques for checking for inconsistencies in the equation system such as checking if the subsystem of linear equations is solvable etc.

7.4 Targeting the Low Number of Initialization Rounds

A2U2 has a secret number of initialization rounds. There are 32 possible choices for the number of initialization rounds that varies from 9 to 126 where each choice is specified by the 5 LSB of the tag's random number, the reader's random number and the 5-bit counter key. In this section, we propose two attacks on A2U2 when 9 initialization rounds are used. The first attack recovers the 5-bit counter key using 2^{14} different state pairs with a specified difference, while the second attack recovers 32 secret key bits and 6 subkey bits using 8 plaintext/ciphertext pairs with a time complexity 2^{38}. Both of the attacks use known plaintexts and chosen IVs.

7.4.1 Recovering the 5-bit Counter Key

The following attack requires for each of the possible 32 initializations, a certain number of state pairs (chosen IVs) with a good difference (sparse characteristic). Under these states we use the ciphertext of a single bit of plaintext that is equal to the first bit of the counter, C_0, at the time when the encryption starts (known plaintext). Then we can distinguish the state pairs corresponding to 9 rounds of initialization by observing a bias in the difference of the first bit of the corresponding ciphertext pairs, $\Delta(Y_0)$, which is equal to the difference $\Delta(B_0)$.

Experiments show that $\Pr(\Delta(B_0) = 0) > 0.7$ for 9 initialization rounds when 2^9 state pairs with differences $\Delta(A) = 10000000000000000$ and $\Delta(B) = 100000100$ are used. The bias is smaller when more initialization rounds are applied. We observe the strong bias in B_0 for 9 rounds because in only 9 rounds the difference cannot propagate through state and does not spread out sufficiently. After having distinguished the 2^9 state pairs corresponding to 9 rounds of initialization, we can consequently find the 5-bit counter key.

7.4.2 Recovering the Master Key Bits

The following attack targets plaintext/ciphertext generated using 9 rounds of initialization (can be obtained using chosen IVs) and exploits the key scheduling used to generate the subkey bits, h_t. The attacker starts by guessing the 26 master key bits used in initializing registers A and B and then at each round he guesses one subkey bit if $C_{t-1} = 0$, or two master key bits if $C_{t-1} = 1$ (since A_{t-2} will then be used to generate h_t) or no bits when all the key bits involved in the generation of the round subkey bit are from the 26 master key bits used in initializing registers A and B.

Table 7.1: List of the master key bits used in the subkey generation of each round, together with the counter value and the number of required guesses. $r \equiv$ round no, $G \equiv$ number of subkey/key bits that are guessed.

r	S_0^t	S_1^t	S_2^t	S_3^t	S_4^t	C_{t-1}	G
0	26	27	28	29	30	0	1
1	31	32	33	34	35	0	1
2	36	37	38	39	40	0	1
3	41	42	43	44	45	1	2
4	46	47	48	49	50	1	2
5	51	52	53	54	55	1	2
6	0	1	2	3	4	1	0
7	5	6	7	8	9	1	0
8	10	11	12	13	14	1	0
9	15	16	17	18	19	1	0
10	20	21	22	23	24	0	0
11	25	26	27	28	29	0	1
12	30	31	32	33	34	0	1
13	35	36	37	38	39	0	1

The cipher is initialized using 9 rounds and then a 5-bit plaintext is encrypted, so in total the cipher runs for 14 rounds. Without loss of generality we assume that the starting key position is 0, so the key bits at positions 0 to 25 are used in initializing registers A and B. Table 7.1 shows the key bits positions used from round 0 to round 13, the value of C_{t-1} and the number of guessed subkey/key bits. The table also shows that we have to guess 6 subkey bits and 6 master key bits plus the 26 master key bits used in initializing the registers. Note that when $6 \leq r \leq 10$, the number of guesses G is zero regardless of the value of C_{t-1} as all the five master key bits involved in generating the corresponding round subkeys belong to the master key bits at positions 0 to 24 (as shown in Table 7.1) which are part of the 26 master key bits which we first guess to initialize registers A and B.

To recover 32 master key bits and 6 subkey bits using plaintexts of length 5-bit, we need only $\lceil \frac{38}{5} \rceil = 8$ plaintext/ciphertext pairs of length 5-bit to find the right key guess. The remaining 24 master key bits can be recovered using a brute force search. Thus, the total complexity of the attack is in the order of 2^{38}. In order for the above attack to work we need to find only 8 plaintext/ciphertext pairs whose initial state pairs are initialized with position 0 as a starting key position and are generated using 9 initialization rounds which we can easily find using the 5-bit counter key recovered in the previous attack.

7.5 Final Remarks

7.5.1 Necessary Changes & Possible Improvements

In this section, we summarize the weaknesses of the published version of A2U2 [46] and propose some possible improvements that would prevent the attacks described in this paper. However, these possible improvements may be prone to other types of attacks, and a full re-evaluation of the cipher would need to be performed in order to assess the strenghts of these potential changes. The following weaknesses make the above attacks possible:

- The fact that the adversary knows the counter state at every instant facilitates cryptanalysis. In particular, it significantly simplifies all algebraic expressions in the cipher, bringing the algebraic degree down to 2 or even 1. Note that to fix this problem, a completely new mechanism for key/IV setup has to be developed. In addition, the size of the counter register has to be increased significantly, since with the current register size, the adversary can just guess the counter starting state which only contributes a factor of 2^7 to the attack complexity. Furthermore, the potentially low number of nine clock cycles to initialize the cipher is not sufficient to ensure a full mixing of the IV bits inside the registers, giving the attacker some valuable information regarding the untouched bits. There are several possible and complementary ways to fix these weaknesses:

  1. In order to increase the number of rounds during initialization, the counter can be set to an all-ones sequence and run for 127 clock cycles ($2^7 - 1$) until it reaches an all-ones sequence again. This would ensure a proper mixing of both registers bits.
  2. At the end of the initialization phase, the counter value could be replaced by the 7 LSB of the secret key, reserved for this sole purpose. The 2 LSB that are fixed in the original cipher description for initialization would no longer be required since this new operation would occur after the 127 initialization rounds. This would prevent an attacker from knowing the exact sequence of bits generated by the counter.
  3. However, this second measure does not solve the problem that the attacker just can guess the initial 7 bits of the counter. As mentioned above, increasing the length of the register would solve this issue, but it would be at the expense of numerous additional gates (which contradicts the design principles of this cipher). A more reasonable solution, in terms of gates, would be to XOR one (or several) external bit(s) to the feedback function of the register. These bits could for example be the output of the key scheduling mechanism or a bit from one of the nonlinear registers.

In this way, the length of the register period would be significantly increased at a very low cost.

- The original design mixes random values both from the reader and the tag to avoid replay and man-in-the-middle attacks [46], which are of common concern in RFID systems. However, the fact that the adversary has an influence (up to total control) on the IV is a serious weakness for a stream cipher. In particular, it must not be possible for the attacker to repeat the IV (nonce-respecting adversary [100]). Thus, it is common practice for stream ciphers that the sender chooses the IV, e.g. as a counter or random value. This design criterion should be adhered to. The registers' IVs may be set as an XOR operation of the tag's pseudo-random number and the secret key.

- The key size itself should be increased, since a total key length of 56 bit is not strong enough for modern standards. Due to the structure of the key scheduling mechanism, the key size needs to be relatively prime with 5 in order to obtain the longest rotation possible before reusing the same 5 key-bits. In addition, 7 bits need to be reserved for the counter (after initialization). A minimum of 88 bits (81 + 7) would seem to be a more reasonable choice. Given that the primary target of the printed tag encryption protocol is to encrypt an EPC of 96 bits [49], having a key size longer than this value would be at the compromise of the area available for the cryptosystem.

As mentioned earlier, when these weaknesses are fixed, a full re-evaluation of the cipher is necessary. Note that in particular, the output unit leaks information (in form of a correlation) about the inner state. Whether or not this information can be used for an attack depends on the inner workings of the cipher. No simple answers can be given here without a full specification of the new design.

Even though having short-size registers and building blocks will necessarily affect the strengh of a cipher, design decisions have to keep in mind the need of designing a cipher small enough to be integrated in printed electronics, where every additional gate counts[2]. This particular target may have some practical sides regarding the amount of data one can collect. According to the specifications of the ISO 14443 Standard [68], with a communication session of 400ms between tag and reader, it would take more than 5 years to collect 2^{36} bits of ciphertext from a single tag, and approximately 6 months to collect 2^{39} bits of ciphertext from 100 tags using the same key. This number could be a limiting factor to break an improved version of A2U2, where no practical attack could be performed with less than 2^{39} bits of known plaintext/ciphertext.

[2]Every additional register in the design implies an increase of (at least) 6.25 GE. Increasing the size of a static key (which is the case for passive RFID tags) comes at a lower cost of 1GE/bit. Moreover, it is often assumed that some space is reserevd for the key inside the device, therefore coming at practically no additional cost

7.5.2 Conclusion

In this chapter we presented several attacks on the lightweight stream cipher A2U2 which all constitute practical breaks of the cipher.

A2U2 is designed for IC printing. To keep the area small, short registers for the inner state are used. A new output generator has been developed to increase the security of the cipher. This output function works similar to the shrinking generator but it overcomes the problem of the irregular output of ciphertext bits by outputting 'dummy' ciphertext bits such that attacker does not know at which positions of the ciphertext plaintext bits were used.

We show that using only two chosen plaintexts, the cipher can be broken in a second by first recovering the sequence which is used to produce the ciphertext and afterwards determine the sequence which controls when plaintext bits are used.

Furthermore, we propose a guess-and-determine attack. We guess the positions where the plaintext bits are used and set up a non-linear equation system which can be solved, e.g., using Gröbner basis techniques. With this approach we can determine the key with a complexity of 2^{49} guesses using a known plaintext of length less than a hundred bits.

We also investigated the possibilities of a chosen-IV attack. Choosing the IV allows us to introduce a difference in the initial state of the register and keeping the counter constant at the same time. Using a differential-style attack, we can identify a bias in the difference of the first ciphertext bit when only 9 initialization rounds are used. Thus, we can recover the counter.

When only 9 rounds of initialization are used, we can recover the master key using only 2^{38} computational steps. We find 32 master key bits by guessing depending on the counter either master key bits or the auxiliary value h_t. The remaining master key bits can be found by exhaustive search.

We conclude our analysis by pointing out some of the most severe weaknesses of the cipher. The biggest and most obvious weakness is that the counter value at the beginning of the encryption is known. However, any change made to the cipher demands a re-analysis of the security. In the current state the cipher can be considered as badly broken.

Chapter 8

Conclusion

In this thesis, we evaluated the security of some lightweight ciphers. Specifically, we studied the security of PRESENT-like ciphers, namely the block cipher PRESENT, the core permutation of the hash function SPONGENT which can be seen as a block cipher with a zero round key and the block cipher PRINTcipher. We also analyzed the security of the stream cipher A2U2. In the following we summarize the results of this thesis and suggest some possible future work.

An approach for estimating the probability of low-weight differential and linear approximations in PRESENT-like ciphers has been provided in Chapter 4. This approach could also be used for any cipher allowing low-weight differential and linear characteristics. The approach used sparse submatrix of the correlation matrix with input and output masks having Hamming weight at most 4 to find better linear approximations in SPONGENT and PRESENT. One possible future work could be to combine this approach with the branch and bound algorithm in order to add the good probabilities that might come from differential or linear characteristics containing intermediate characteristics with Hamming weight more than 4. Another possible future work could be to use parallel computing to construct matrices with input and output entries with Hamming weight more than 4 to see whether this would improve the linear attacks on PRESENT. Other possible improvements to speed up the submatrix construction when considering input and output entries with Hamming weight more than 4 could be considering only calculating the correlations of those inputs which activate at most 2 Sboxes in the submatrix while setting the correlations of the other inputs to zero.

An interesting design topic could be to fix the PRESENT-like involution cipher PUFFIN – which is not secure against both differential and linear cryptanalysis [20, 82] – by finding the right combination of Sbox and bitwise permutation where both of them are involution that is resistant against differential and linear cryptanalysis for a number of rounds that is equivalent to the number of rounds used by PRESENT. It would be interesting to see if such a combination exists as it was shown in [53] that there is no Sbox with involution satisfying the PRESENT Sbox criteria and it was also shown in [82] that there is no permutation with involution satisfying the PRESENT permutation criteria – that is being optimal in the sense

that full dependency is achieved in a minimum number of rounds. Note that this topic was previously proposed in [20]. We believe that the submatrix approach would speed up the search for the right combination of Sbox and permutation.

In Chapter 4, we also studied the effect of key scheduling on PRESENT-like ciphers. The main result of this study is in showing that the distribution of linear approximations with input and output masks in PRESENT with identical round keys has a larger variance than those in PRESENT with the standard key scheduling of PRESENT (or independent round keys). This shows that a PRESENT variant with identical round keys is less secure against linear cryptanalysis than PRESENT with the standard key scheduling of PRESENT (or independent round keys). More precisely, the fraction of keys with a squared correlation larger than 2^{-53} is 33.7% in the case of identical round-keys but only 1.1% in the case of the standard PRESENT key-scheduling. One possible future work is to study the distribution of linear approximations with input and output masks with Hamming weight more than one in PRESENT with identical round keys and see whether they would have a larger variance compared to input and output masks with weight one.

In Chapter 5, we showed how to mount a differential attack on reduced rounds of PRINTcipher which uses a key dependent layer that seems to complicate differential attacks. One of the two proposed differential attacks recovers the key dependent permutation layer of an r-round PRINTcipher by computing the r-th root of a permutation. The chapter also shows that a secure authentication protocol based on the many r-th roots of a permutation could be designed. As pointed previously, we believe that finding a suitable mixing function for the challenges and the secret key used in the authentication protocol beside a hardware implementation for this protocol are two interesting topics for future work.

In Chapter 6, we presented the invariant subspace attack and shows it relation to the statistical saturation attacks. The invariant subspace attack was successful on the full round PRINTcipher and it found two classes of weak keys. Later the whole classes of weak keys in PRINTcipher was found.

In Chapter 7, we presented several attacks against the stream cipher A2U2. Using two chosen plaintexts we showed that the cipher can be broken in a second. A guess-and-determine known plaintext attack was proposed. We proposed two attacks that exploit the use of low number of initialization rounds. We also pointed out some possible improvements that would make the cipher resistant against the proposed attacks.

Bibliography

[1] Mohamed Ahmed Abdelraheem. Estimating the probabilities of low-weight differential and linear approximations on PRESENT-like ciphers. In *The 15th Annual International Conference on Information Security and Cryptology, ICISC*, 2012. to appear. v

[2] Mohamed Ahmed Abdelraheem, Martin Ågren, Peter Beelen, and Gregor Leander. On the distribution of linear biases: Three instructive examples. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2012. vi, 99, 100

[3] Mohamed Ahmed Abdelraheem, Julia Borghoff, Erik Zenner, and Mathieu David. Cryptanalysis of the light-weight cipher A2U2. In Liqun Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2011. vi

[4] Mohamed Ahmed Abdelraheem, Gregor Leander, and Erik Zenner. Differential cryptanalysis of round-reduced PRINTCIPHER: Computing roots of permutations. In Joux [69], pages 1–17. vi

[5] M. Ågren. *On Some Symmetric Lightweight Cryptographic Designs*. PhD thesis, Sweden, Lund University, 2012. 100

[6] Martin Ågren and Thomas Johansson. Linear cryptanalysis of printcipher - trails and samples everywhere. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 114–133. Springer, 2011. 51

[7] Martin Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. *IACR Cryptology ePrint Archive*, 2012:401, 2012. 65, 66

[8] Scott Annin and Trenton Jansen. On kth roots in the symmetric and alternating groups. *Pi Mu Epsilon Journal*, 12(10):581–589, 2009. 78

[9] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. In Mangard and Standaert [85], pages 1–15. 45

[10] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2004. 25

[11] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *FSE*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998. 16

[12] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In Stern [108], pages 12–23. 19

[13] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990. 17

[14] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991. 13, 17

[15] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2004. 25

[16] Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999. 36, 68

[17] Céline Blondeau and Benoît Gérard. Links between theoretical and effective differential probabilities: Experiments on present. *IACR Cryptology ePrint Archive*, 2010:261, 2010. 66

[18] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: Theory and practice. In Joux [69], pages 35–54. 17, 46, 55

[19] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: Theory and practice. In Joux [69], pages 35–54. 18

[20] Céline Blondeau and Benoit Gérard. Differential Cryptanalysis of PUFFIN and PUFFIN2, 11 2011. 54, 112, 113

[21] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varici, and I. Verbauwhede. SPONGENT: The design space of lightweight cryptographic hashing. *Computers, IEEE Transactions on*, PP(99):1, 2012. 33, 56

[22] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. SPONGENT: A lightweight hash function. In Preneel and Takagi [98], pages 312–325. 39, 40, 55, 56, 62

[23] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007. 38, 40, 60

[24] Andrey Bogdanov, Gregor Leander, Kaisa Nyberg, and Meiqin Wang. Integral and multidimensional linear distinguishers with correlation zero. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 244–261. Springer, 2012. 32

[25] Andrey Bogdanov and Vincent Rijmen. Zero-correlation linear cryptanalysis of block ciphers. *IACR Cryptology ePrint Archive*, 2011:123, 2011. 32, 64

[26] Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 29–48. Springer, 2012. 32

[27] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Cryptanalysis of PRESENT-like Ciphers with Secret S-Boxes. In Joux [69], pages 270–289. 33, 40, 54

[28] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Slender-set differential cryptanalysis. *Journal of Cryptology*, pages 1–28, 2011. 54

[29] M. L. Buchanan and B. N. Parlett. The uniform convergence of matrix powers. *Numerische Mathematik*, 9(1):51–54, 1966. 99

[30] Stanislav Bulygin and Michael Walter. Study of the invariant coset attack on printcipher: more weak keys with practical key recovery. *IACR Cryptology ePrint Archive*, 2012:85, 2012. 50, 51, 100

[31] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In *Proc. CHES 2009*, volume 5747 of *Springer LNCS*, pages 272–288, 2009. 45

[32] Christophe De Cannière and Bart Preneel. Trivium. In Robshaw and Billet [99], pages 244–266. 4, 19, 44

[33] Anne Canteaut, Claude Carlet, Pascale Charpin, and Caroline Fontaine. On cryptographic properties of the cosets of r(1, m). *IEEE Transactions on Information Theory*, 47(4):1494–1513, 2001. 30

[34] Claude Carlet. Boolean functions for cryptography and error correcting codes. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, 134:257, 2010. 27

[35] Claude Carlet. Vectorial boolean functions for cryptography. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, (134):398–471, 2010. 22

[36] Qi Chai, Xinxin Fan, and Guang Gong. An ultra-efficient key recovery attack on the lightweight stream cipher A2U2. http://eprint.iacr.org/2011/247, 2011. Version published: 20110518:133751 (posted 18-May-2011 13:37:51 UTC). 51

[37] Huiju Cheng, Howard M. Heys, and Cheng Wang. PUFFIN: A novel compact block cipher targeted to embedded digital systems. In Luca Fanucci, editor, *DSD*, pages 383–390. IEEE, 2008. 54

[38] Joo Yeon Cho. Linear cryptanalysis of reduced-round present. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010. 28, 46, 55, 61

[39] Baudoin Collard and François-Xavier Standaert. A statistical saturation attack against the block cipher PRESENT. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2009. 29, 31, 45, 55, 62, 71

[40] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003. 36

[41] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In Preneel [97], pages 275–285. 22

[42] Joan Daemen, Michael Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON. http://gro.noekeon.org/Noekeon-spec.pdf, 2000. 96

[43] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001. 35

[44] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002. 22

[45] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *IACR Cryptology ePrint Archive*, 2005:212, 2005. 31, 64, 65, 66, 67

[46] M. David, D.C. Ranasinghe, and T. Larsen. A2U2: A stream cipher for printed electronics RFID tags. In *IEEE International Conference on RFID*, pages 176 –183, 2011. 42, 102, 109, 110

[47] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. 18

[48] Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In Joux [69], pages 167–187. 19

[49] EPC Global. *EPC Class 1 Generation 2 UHF Air Interface Protocol Standard "Gen 2"*, 2008. 110

[50] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for rfid systems using the aes algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004. 37, 38

[51] A.S. Fraenkel and Y. Yesha. Complexity of solving algebraic equations. *Information Processing Letters*, 10(4):178–179, 1980. 36

[52] M. Gomathisankaran and R.B. Lee. Maya: A novel block encryption function. In *Proceedings of International Workshop on Coding and Cryptography*, 2009. 33, 54

[53] Zheng Gong, Svetla Nikova, and Yee Wei Law. Klein: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011. 112

[54] T. Good and M. Benaissa. Hardware performance of estream phase-iii stream cipher candidates. In *In SASC*, 2008. 45

[55] Anja Groch, Dennis Hofheinz, and Rainer Steinwandt. A practical attack on the root problem in braid groups. In *Algebraic methods in cryptography*, volume 418, pages 121–132. American Mathematical Society, 2006. 78

[56] Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In Rogaway [101], pages 222–239. 45

[57] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The led block cipher. In Preneel and Takagi [98], pages 326–341. 45

[58] Carlo Harpes, Gerhard G. Kramer, and James L. Massey. A generalization of linear cryptanalysis and the applicability of matsui's piling-up lemma. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT*, volume 921 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 1995. 25

[59] Philip Hawkes and Luke O'Connor. Xor and non-xor differential probabilities. In Stern [108], pages 272–285. 16

[60] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Robshaw and Billet [99], pages 179–190. 4, 44

[61] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. 8, 10

[62] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis of reduced round serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2008. 25, 26, 27

[63] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional extension of matsui's algorithm 2. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2009. 25, 28, 61

[64] Miia Hermelin and Kaisa Nyberg. Linear cryptanalysis using multiple linear approximations. Cryptology ePrint Archive, Report 2011/093, 2011. 60, 61

[65] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001. 37

[66] Ecrypt II. Ecrypt ii yearly report on algorithms and keysizes. 2011-2012. 8

[67] ISO/IEC 29192-2:2012. Information technology Security techniques Lightweight cryptography. Part 2: Block ciphers. 2012. 54

[68] ISO/IEC 14443-2 Standard. *Identification cards - Contactless integrated circuit cards - Proximity cards - Part 2: Radio frequency power and signal interface*, 2010. 110

[69] Antoine Joux, editor. *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*. Springer, 2011. 114, 115, 116, 118

[70] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear cryptanalysis using multiple approximations. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. Springer, 1994. 25

[71] Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005. 37

[72] Ferhat Karakoç, Hüseyin Demirci, and A. Emre Harmanci. Combined differential and linear cryptanalysis of reduced-round printcipher. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2011. 51

[73] Miroslav Knežević. Personal Communication, 2011. 89

[74] Lars Knudsen. Deal - a 128-bit block cipher. In *NIST AES Proposal*, 1998. 19

[75] Lars R. Knudsen. *Block Ciphers - Analysis, Design and Applications*. PhD thesis, Denmark, Aarhus University, 1994. 6

[76] Lars R. Knudsen. Truncated and higher order differentials. In Preneel [97], pages 196–211. 17, 18

[77] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-printing. In Mangard and Standaert [85], pages 16–32. 39, 41, 50, 75

[78] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. Printcipher: A block cipher for IC-Printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010. 98

[79] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information security and cryptography. Springer, 2011. 6, 17, 35

[80] X. Lai. Higher order derivatives and differential cryptanalysis. *KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE*, pages 227–227, 1994. 18

[81] Xuejia Lai and James L. Massey. Markov ciphers and differentail cryptanalysis. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991. 15, 16, 17, 33, 55

[82] Gregor Leander. On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In Paterson [95], pages 303–322. 29, 30, 31, 39, 46, 54, 55, 97, 112

[83] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of printcipher: The invariant subspace attack. In Rogaway [101], pages 206–221. vi, 91

[84] Jesús Leaños, Rutilo Moreno, and Luis M. Rivera-Martínez. A note on the number of m-th roots of permutations. *Arxiv preprint arXiv:1005.1531*, 2010. 78, 79

[85] Stefan Mangard and François-Xavier Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*. Springer, 2010. 114, 120

[86] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993. 19, 21

[87] Mitsuru Matsui. On correlation between the order of s-boxes and the strength of des. In Santis [103], pages 366–375. 33, 55

[88] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. 4, 12

[89] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In Paterson [95], pages 69–88. 37

[90] Kaisa Nyberg. Linear approximation of block ciphers. In Santis [103], pages 439–444. 31

[91] Luke O'Connor and Jovan Dj. Golić. A unified markov approach to differential and linear cryptanalysis. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 387–397. Springer, 1994. 55

[92] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003. 11

[93] Kenji Ohkuma. Weak keys of reduced-round PRESENT for linear cryptanalysis. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected*

Areas in Cryptography, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2009. 45, 55, 60, 67

[94] R. Oldenburger. Infinite powers of matrices and characteristic roots. *Duke Mathematical Journal*, 6(2):357–361, 1940. 99

[95] Kenneth G. Paterson, editor. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011. 121

[96] A. I. Pavlov. On the number of solutions of the equation $x^k = a$ in the symmetric group S_n. *Mathematics of the USSR-Sbornik*, 40(3):349–362, 1981. 79

[97] Bart Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995. 117, 120

[98] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011. 115, 118, 123

[99] Matthew J. B. Robshaw and Olivier Billet, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008. 116, 119

[100] P. Rogaway. Nonce-based symmetric encryption. In *Proc. FSE 2004*, volume 3017 of *Springer LNCS*, pages 348–359, 2004. 110

[101] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011. 118, 121

[102] Y. Saad. *SPARSKIT: A basic tool kit for sparse matrix computation*. Research Institute for Advanced Computer Science, NASA Ames Research Center, 1990. 59

[103] Alfredo De Santis, editor. *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*. Springer, 1995. 121

[104] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *J. Cryptology*, 21(1):131–147, 2008. 17, 25

[105] Adi Shamir. SQUASH - a new mac with provable security properties for highly constrained devices such as RFID tags. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2008. 37, 88

[106] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949. 13

[107] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Preneel and Takagi [98], pages 342–357. 45

[108] Jacques Stern, editor. *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*. Springer, 1999. 115, 119

[109] Meiqin Wang. Differential cryptanalysis of reduced-round PRESENT. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2008. 45, 55

[110] Herbert S. Wilf. *Generatingfunctionology*. Academic Press, 1993. 77, 78

[111] Dengguo Feng Zhenqing Shi, Xiutao Feng and Chuankun Wu. A real-time key recovery attack on the lightweight stream cipher a2u2. In *CANS*, Lecture Notes in Computer Science. Springer. 52